RingStack Help

Everything the console and `ringstack-cli` can do, and how to use it.

CONTENTS

1. What RingStack is
2. Installing & getting started
3. Extensions & phones
4. Trunks & numbers
5. Call routing
6. Queues & ring groups
7. Voicemail, conferencing & parking
8. Text messages
9. Dashboard, logs & reports
10. AI & integrations
11. API & automation
12. Users, teams & access
13. Security & system administration
14. Email delivery & support access relays
15. Licensing
16. Multi-tenant & the Global Admin console
17. RingStack Central (fleet management)
18. High availability
19. Backup & restore
20. ringstack-cli reference
21. Classic vs. Guided navigation
22. Guided tour
23. Troubleshooting & FAQ
24. Getting more help

What RingStack is

A self-hosted phone system, built on Asterisk, with everything managed from this one console.

RingStack replaces a traditional office phone system (a PBX) — it registers desk phones and softphones as extensions, connects to your carrier(s) over SIP trunks, and routes calls between the two based on rules you set up here: inbound routes, IVR menus, ring groups, queues, and time conditions. Everything that reaches Asterisk — dialplan, PJSIP endpoints, trunk registrations — is generated automatically from what you configure in this console; you never hand-edit Asterisk config files directly.

Two install shapes exist. A **classic** install runs one company on one box — everything in this document applies directly. A **multi-tenant** install runs several separate companies on one box, each fully isolated with their own extensions/trunks/routes/branding, managed through a separate Global Admin login — see Multi-tenant & the Global Admin console for how that changes things.

Installing & getting started

From a fresh Ubuntu/Debian box to a working system.

Classic install (one company, one box)

Run the installer as root on a fresh box:

```
curl -fsSL https://raw.githubusercontent.com/ramman1169/ringstack/main/install/install.sh | sudo bash
```

or, from a cloned copy of the repository:

```
sudo ./install/install.sh
```

This installs and wires together Asterisk (PJSIP realtime over ODBC/MySQL), MySQL, the Node backend, nginx (reverse proxy + TLS via certbot), and the systemd service that runs the backend — a complete, working stack when it finishes. With no flags, this is a classic single-tenant install. See `sudo ./install/install.sh --help` for the full flag list (the piped one-liner above has no argv to pass flags through — use the matching `RS_*` environment variable instead, e.g. `RS_MULTI_TENANT=1 RS_INSTALL_ROLE=bootstrap curl ... | sudo bash`).

Once installed, create the first admin account:

```
ringstack-cli create-admin --email you@yourcompany.com --password 'a-real-password'
```

Sign in at the box's own address and you're in the console described throughout the rest of this document.

Multi-tenant install (several companies, one box)

Pass `--multi-tenant` plus a role — there is no bare `--multi-tenant` mode:

FLAG	WHAT IT MEANS
`--multi-tenant --bootstrap`	The **first** node of a fresh multi-tenant install (or a standalone multi-tenant box) — walks through licensing and creating the Global Admin account interactively during install.
`--multi-tenant --secondary`	This box is joining an **already-existing** multi-tenant pair — skips the Global Admin prompt (a joining node inherits everything via replication) and offers to consume a join token generated on the existing primary.
`--is-hosted`	Arms a one-shot de-clone step so every disk clone of this install regenerates its own secrets (SESSION_SECRET, DB/AMI passwords, SIP credentials, API keys, TLS cert) on its first real boot instead of sharing the template's. Set this on a "golden image" VM you intend to clone per customer — never on a normal, already-personalized install.

`--bootstrap` and `--secondary` are mutually exclusive and both require `--multi-tenant`. Multi-tenant mode requires a license with the `multitenant` feature — see Licensing — the bootstrap flow walks you through applying one.

After a multi-tenant bootstrap install, create tenants with `ringstack-cli create-tenant` (see the CLI reference) or through the Global Admin console (see Multi-tenant & the Global Admin console).

Keeping it up to date

`ringstack-cli update` pulls the latest release for your configured channel (stable or edge), applies any new database migrations, redeploys, and restarts — run `ringstack-cli update --check` first to see what would change without applying it. Security-critical fixes are free forever and never gated behind a license or subscription.

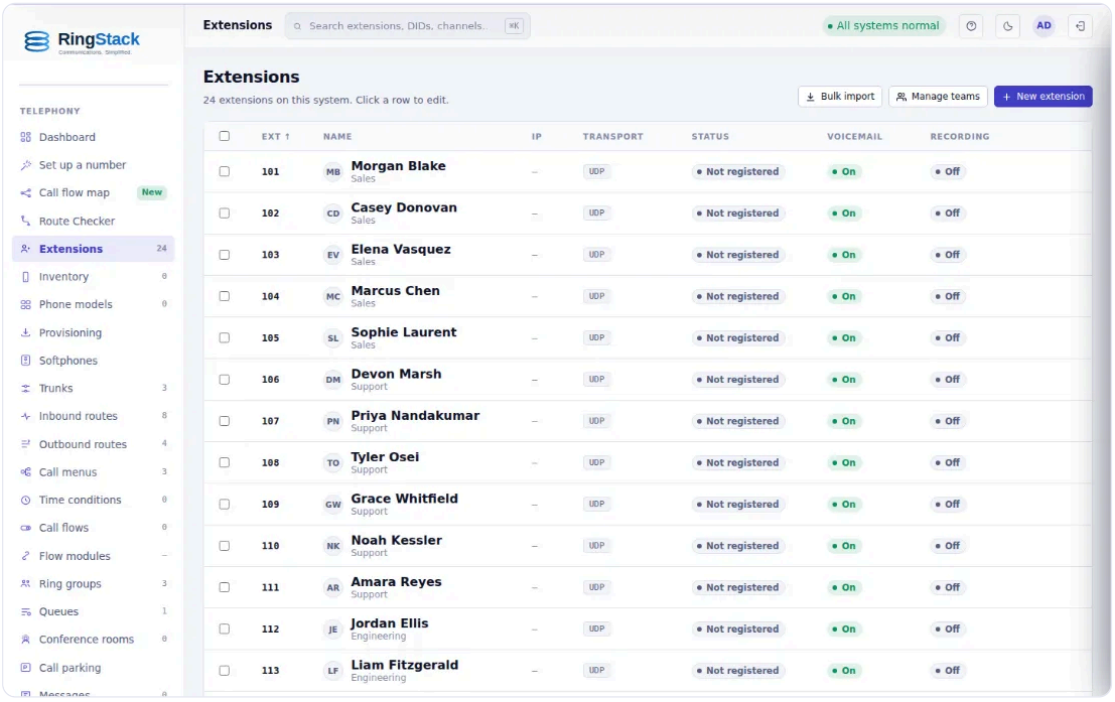
Extensions & phones

The people and devices on your phone system.

Extensions

Each person or line on the system — a short internal number, a display name, and its own SIP credentials. Bulk-import a whole list from a CSV, or add one at a time. Hot-desking lets an agent temporarily claim a shared phone as their own extension for a shift, and call barging/monitor let a supervisor listen in or join a live call. A single extension can have more than one live registration at once — a desk phone, the browser softphone, and the mobile app each register as their own separate device under the same extension number, so all of them can ring together.

Extensions



The Extensions list. Click any row to edit it.

Phones

The physical desk phones assigned to extensions — reboot or resync one remotely once it's registered, or add a new one by MAC address.

`Inventory → Phones`

Activation codes — set up a phone without its MAC address

Pick one extension (or several at once) plus the phone's vendor/model, and get a short 6-digit code per extension instead of having to hunt down and type in each phone's MAC address by hand. Give the code to whoever's racking the phone — on a fresh, unclaimed unit, just picking up the handset auto-dials straight into a short prompt for it; if it doesn't auto-dial, dial `*7777` by hand instead. Entering the right code creates the phone's real inventory record automatically, bound to that extension — nothing to reconcile afterward. Works over HTTPS provisioning (not TFTP) only. Auto-dial is hardware-confirmed for Yealink and Grandstream's WP-series; Polycom, other Grandstream lines, and Sangoma's S-Series are best-effort and fall back cleanly to the `*7777` manual dial-in if the auto-dial field turns out to be wrong for a given firmware. Sangoma's P-Series doesn't auto-dial at all on a generic (non-Sangoma-UC) system — `*7777` is the normal path there, not a fallback.
Some phones (for example the Sangoma P325) don't send a caller name when they dial, so RingStack can't read their MAC from the call. In that case it matches the call to the phone that just fetched its setup file from the same public IP address. If more than one unclaimed phone is behind that address, it can't tell them apart and the code is rejected — activate those one at a time.
Codes are shown once, at the moment you generate them — there's no way to look a code back up afterward, so download the PDF or email it to the installer right away. The PDF (or email) is fully branded and opens with a plain "how to use a code" walkthrough, so it also works handed to a non-technical office manager, not just a tech.

`Inventory → Phones → Generate activation code`

Phone models

Turn on the vendors and models you actually use — enabled models show up in Phones → Add a phone. Auto-provisioning config is generated for Yealink, Grandstream, and Cisco; Poly can be tracked here but won't get a config file yet.
Sangoma and Grandstream models also have a **Config template** choice (Sangoma P-series or S-series, Grandstream GRP/WP). It decides which provisioning template the model uses, independent of what you name it — so you can call a model "Front Desk Phone" and it still gets the right config. Leave it on *Auto-detect* and RingStack guesses from a model number in the name, as before. Changing it rebuilds the configs and tells you how many phones need a push.

`Phone models`

Firmware management

Upload a firmware file directly, or just paste the vendor's download URL and let RingStack fetch it — a .zip gets auto-extracted and its docs pulled out automatically, or point a separate field at the docs on their own. Every version you upload for a model is kept in a real history, so pushing a phone back to a previous version is a normal rollback, not a re-upload. With AI integration on, an uploaded config gets a quick AI syntax check before it's offered for push. Actually pushing firmware out to registered phones is a Pro feature; uploading, viewing versions, and rolling back stay available either way.

`Phone models → Firmware`

Provisioning

Global auto-provisioning settings — TFTP and HTTPS both serve the same generated per-phone config, so a new phone comes up already configured once it's plugged in.

`Provisioning`

Softphones

Which extensions can take calls right in the browser, and who's currently signed in. The same account also signs into the self-service Portal and, once set up, the mobile app (RingStack Connect).

`Softphones`

Feature codes

Star codes any phone can dial with no extra setup — call forward, DND, hot-desk login/logout, call monitor — turn individual ones off if you'd rather that code stay free for something else, or change the code itself if it collides with something your carrier already uses.

`Feature codes`

Trunks & numbers

How calls get on and off your phone system.

Trunks

Your SIP carrier connections and their live registration state — this is the line that actually carries calls to and from the outside world.

`Trunks`

DIDs

Every phone number your account owns, whether it's actively routed somewhere right now or sitting in reserve — a DID can be reachable via more than one trunk, and assigning one just creates its inbound route.

`Inventory → DIDs`

Paired systems

Real inter-PBX SIP trunking to another RingStack install — dial a prefix plus their extension to reach it directly, no separate trunk needed. Manual mutual setup: enter the same shared secret on both systems.

`Paired systems`

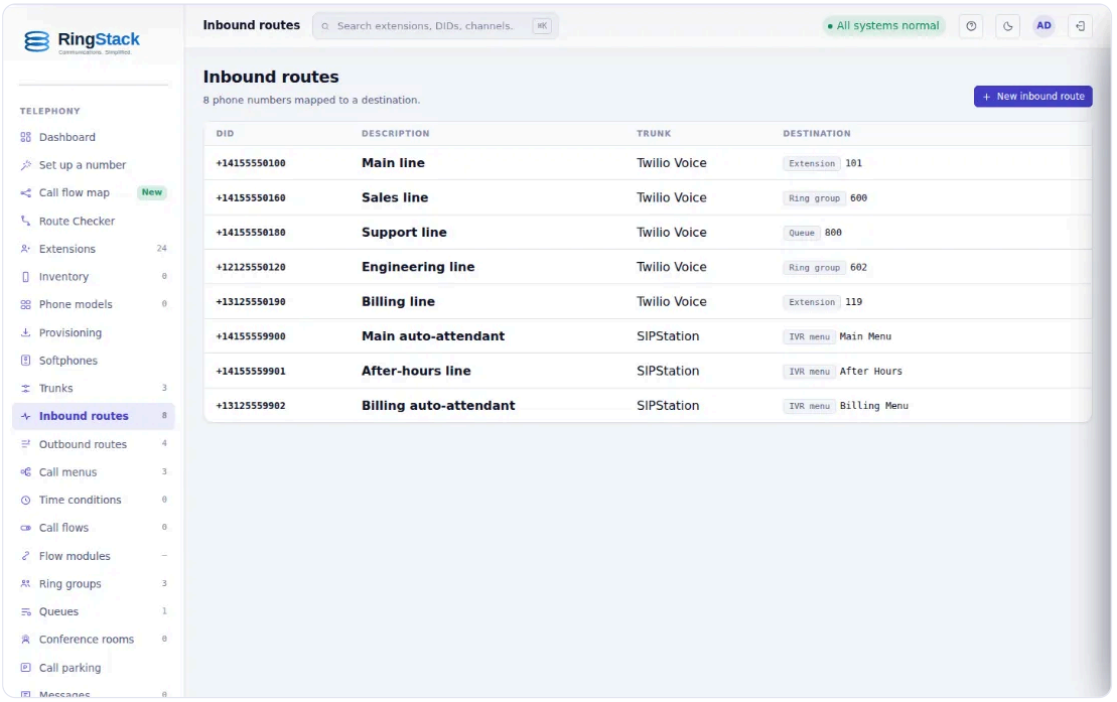
Call routing

How an inbound call gets from a ringing number to the right person — and how an outbound call picks which trunk carries it.

Inbound routes

Where each of your numbers actually goes — an extension, an IVR menu, a ring group, or anywhere else with a destination.

`Inbound routes`



Inbound routes map each phone number to a destination.

Outbound routes

Which trunk carries an outgoing call, and who's allowed to dial what. Routes are tried in order — the first match wins.

`Outbound routes`

Call menus (IVR)

What callers hear — "For Sales, press 1" — and where each key press sends them, including further into another menu.

`Call menus`

Time conditions

Route calls differently by day and time — one destination inside the window, a different one outside it. Business-hours evaluation uses your configured company timezone (Settings → Company), not the server's own OS clock — see why that matters if it's ever wrong.

`Time conditions`

Call flows

A manual override switch between two destinations (day/night, open/closed) — flip it from any phone with a control-number dial, or flip it right here.

`Call flows`

Flow modules

A single named, reusable routing step, including its own Call Flow Canvas node type. Build it once, then point any number of time conditions, IVR menus, or ring groups at it — editing the module changes it everywhere it's used.

`Flow modules`

Call flow map

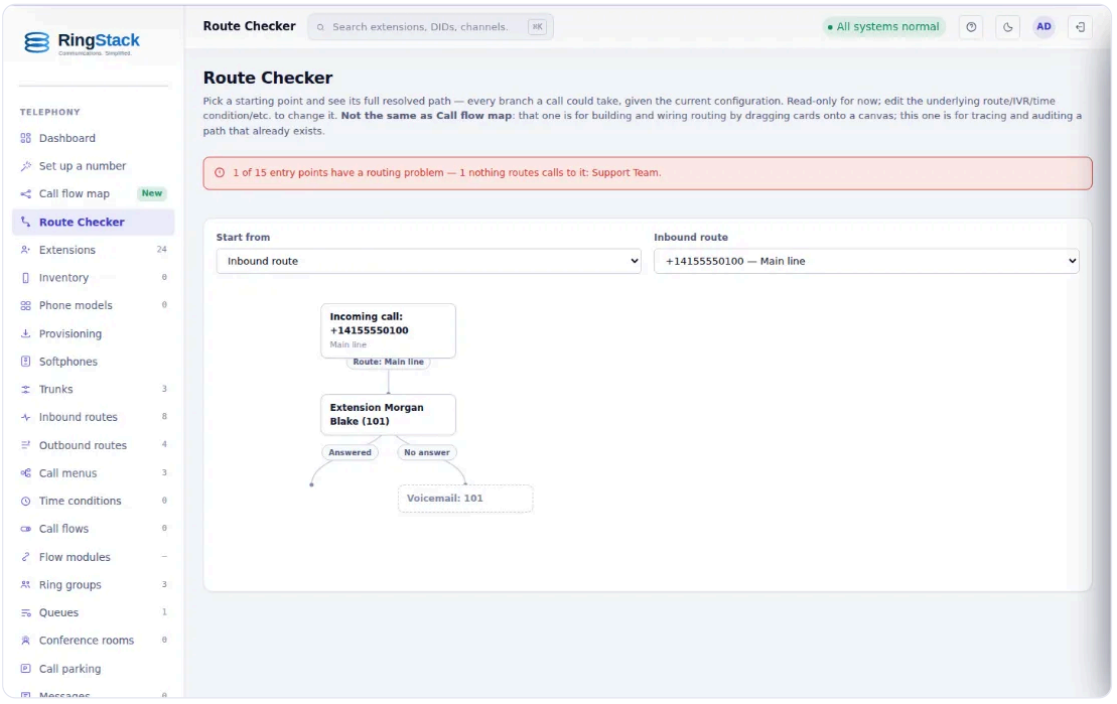
Every inbound route, time condition, call flow, IVR menu, flow module, ring group, and queue, wired together on one visual canvas — build and rewire routing here by dragging cards.

`Call flow map`

Route Checker *(Beta)*

Pick a starting point and see its full resolved path — every branch a call could actually take, given the current configuration. Read-only — a diagnostic/safety tool for tracing a path that already exists, not for building one (that's Call flow map).

`Route Checker`

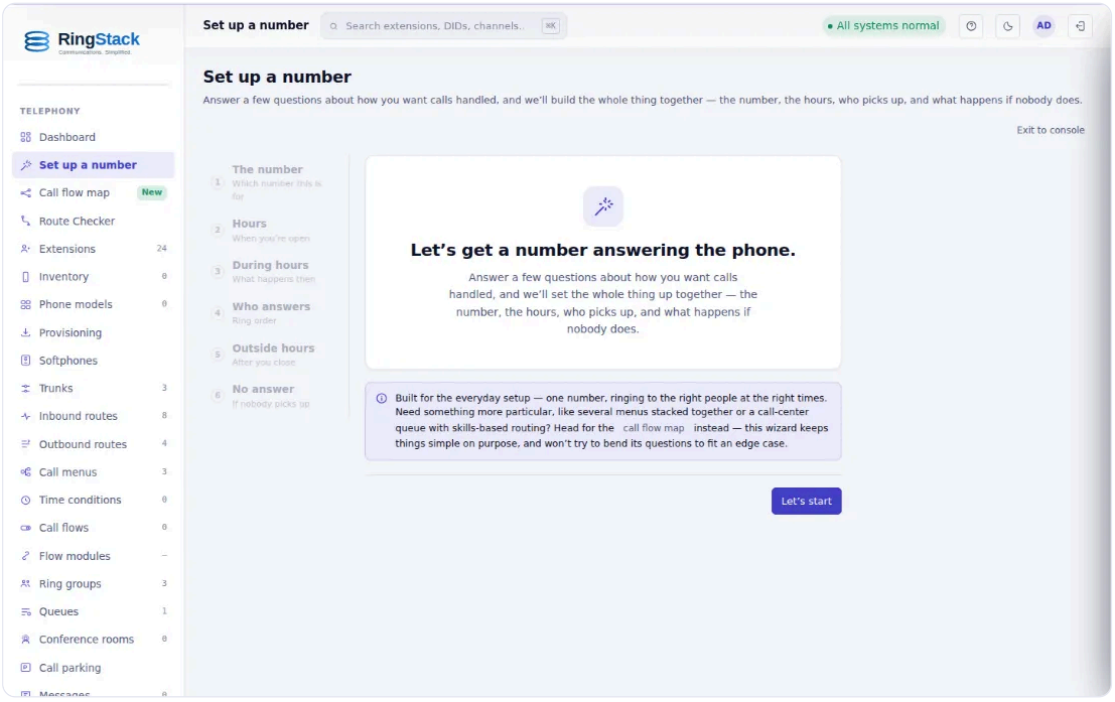


The Route Checker tracing one inbound number end to end, including where an unanswered call ends up.

Set up a number

A guided wizard — answer a few questions about how you want calls handled, and it builds the number, hours, and destinations for you.

`Set up a number`



The **Set up a number** wizard opens with its six steps listed on the left.

Queues & ring groups

Getting an inbound call to more than one person at once.

Ring groups

Ring several phones for one call — no waiting line, no agents logging in or out. Whoever answers first gets it.

`Ring groups`

Queues

Ring several agents at once and hold callers fairly in line when everyone's busy — the call-center version of a ring group. Agents can log themselves in or out from the Portal instead of an admin editing the roster, and a queue can offer a callback so a caller on hold can press 1 and get called back instead of waiting. The Wallboard shows live status; Reports has queue performance and an SLA-aware scorecard.

`Queues, Wallboard`

Wallboard

Live queue status on its own full screen — agent state, calls waiting, longest wait — customizable the same way the Dashboard is (add/remove/reorder tiles). Pop it out to its own window, or set up an unattended kiosk display (a wall-mounted screen, a Raspberry Pi) with a dedicated display link that never logs itself out and needs no one sitting at it to keep the session alive.

`Wallboard`

Voicemail, conferencing & parking

Voicemail

Every mailbox with voicemail turned on, in one place — reset a PIN, change where messages get emailed, or find a specific message.

`Voicemail`

Conference rooms

Shared dial-in bridges anyone can join with a PIN — no scheduling needed per call.

`Conference rooms`

Call parking

Put a call on hold in a shared slot so anyone can pick it up from a different phone. Asterisk's default lot (dial 700, slots 701–720) always exists.

`Call parking`

Audio library

Upload once, use anywhere — an IVR greeting, a queue's hold music, or a mailbox's voicemail greeting. Everything you upload is converted to the format Asterisk actually needs. Group related clips together, or generate a new one from typed text with an AI voice instead of recording it yourself (needs AI integration turned on — see AI & integrations).

`Audio library`

Text messages

Real SMS, over your existing trunk.

Messages

Sent as SIP MESSAGE (RFC 3428), going over the same trunk connection that already carries your calls — no separate carrier API or webhook to configure. A DID can carry SMS over more than one trunk, and messages both ways show up in the same thread. Your trunk provider has to actually support SMS over SIP for this to work — this sends/receives whatever the trunk is willing to carry, it can't add SMS support a carrier doesn't offer. Extensions with the mobile app can send/receive SMS for their own assigned DID from the app itself.

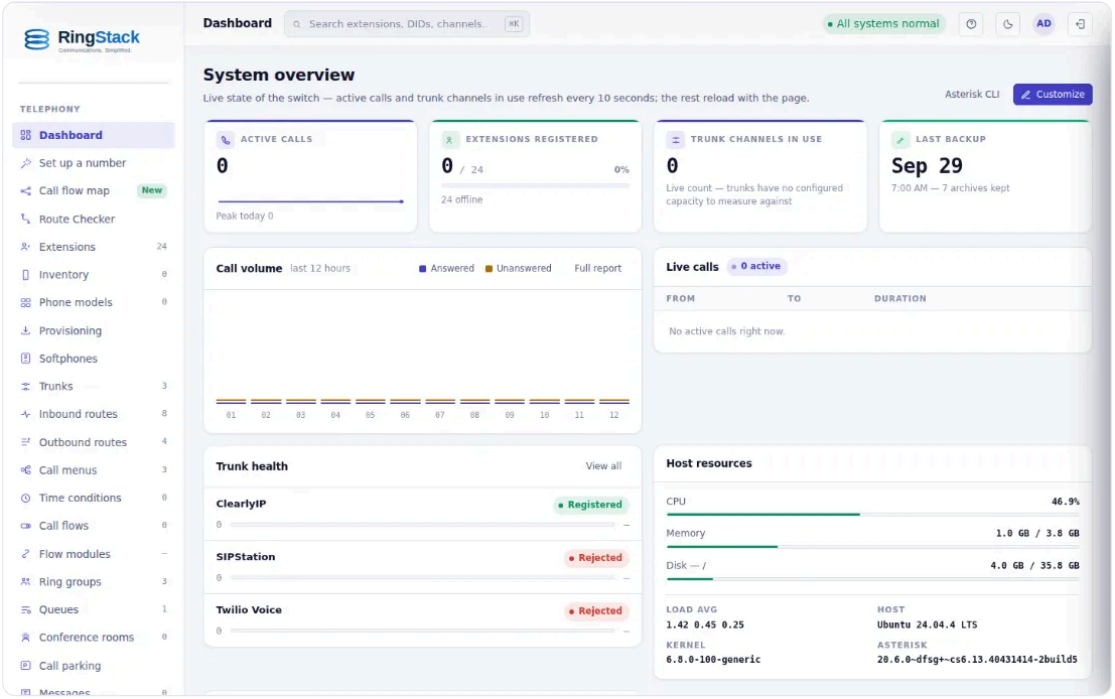
`Messages`

Dashboard, logs & reports

Dashboard

Live state of the switch. Active calls and trunk channels in use refresh every few seconds; the rest reloads with the page. Fully customizable — add, remove, and reorder cards, including a topbar clock showing the current time in your configured company timezone next to the config-sync status pill. The search bar in the topbar finds extensions, DIDs, and live calls by number/name, and also jumps straight to any settings or nav page by name — type "TLS" or "firewall" and pick the result.

`Dashboard`



The Dashboard. Every card can be added, removed or reordered with **Customize**.

Call logs

Every call, searchable by number, date range, and outcome, with recordings where enabled. The outcome says what actually happened: **Answered** (by whom), **Missed** (for example "Hung up in the queue after 2:35" or "Hung up in the menu"), or **Voicemail**; outbound calls show Answered, No answer, Busy or Failed. A caller who hangs up while waiting in a queue or a menu is Missed, even though the phone system technically answered the line. Open a call to see its full path, how long the caller waited, and how it ended. Drill into a specific call for its real SIP signaling trace.

`Call logs`

Reports

How your phone system performs over time — call volume, queue performance, and where calls actually go. Export any report to PDF or CSV on demand, email it to someone, or put it on a recurring schedule (daily/weekly/monthly).

`Reports`

Pro Reports

Deeper analytics on top of the standard reports — an agent performance scorecard, repeat-caller detection, concurrency peaks, and IVR containment (how many callers actually get where a menu sends them vs. drop out). With AI integration turned on, the same data an AI assistant can answer free-form questions about (Ask): it shows which reports it pulled from, can draw a chart from their numbers, and one click saves the question to My reports (named after the question; rename it there), where it can be re-run or scheduled. Plus prose narration of any report.

`Pro Reports`

Fraud detection

Watches live call activity for toll-fraud call-rate spikes, domestic traffic pumping, and any destination you've flagged.

`Fraud detection`

Troubleshoot

Ask an AI assistant to help diagnose a real problem — it can trace the dialplan for a number, check trunk/extension registration, read recent Asterisk log errors, and pull the actual SIP signaling trace for a failed call. Read-only — it recommends fixes, it never changes anything itself. Needs AI integration turned on first.

`Troubleshoot`

AI & integrations

Connecting RingStack to a model provider and to your own systems.

AI integration *(Beta)*

Bring your own key (Anthropic, OpenAI, or Google) — RingStack's backend calls it server-to-server for a few scoped tasks. Nothing runs on the PBX box itself, and no call audio or transcript ever leaves your box unless a specific AI feature you've turned on needs it for that one request. Optionally pick a specific model instead of the provider's default. Once turned on, it powers Troubleshoot, call-journey narration on Call logs/Reports, the Pro Reports Ask assistant, fraud-alert narration, recording transcription, and AI-generated audio-library voice prompts.

`Settings → AI integration`

Webhooks

Fires an outbound HTTP POST (caller ID, DID, a timestamp) mid-call, then keeps going to its own next destination — pick "Fire webhook: <name>" anywhere a destination is chosen (inbound routes, IVR options/timeout, time conditions, ring group/queue no-answer fallback, call flows, custom feature codes, flow modules). Never blocks the call either way, and requests to a private/internal address are refused.

`Webhooks`

API keys

Programmatic access for scripts and integrations that can't sign in interactively. Creating and using API keys requires a license that includes the `pro` feature: without one, requests made with a key are refused (HTTP 402), and after a license expires keys keep working for a 14-day grace period (responses carry an `X-RingStack-License: grace` header) before they stop. Signing in to the console is never affected. Each key can carry its own expiration date and be scoped to only the resources it actually needs, and is rate-limited per key. See API & automation for how to authenticate and what to expect back, and `/api-docs` on your server for the full endpoint reference.

`API keys`

FreePBX import

Migrate extensions, trunks, inbound/outbound routes, ring groups, and queues from a real FreePBX "Backup and Restore" archive (.tar.gz). Nothing is written until you review the preview and confirm — every record is created through the same real API a manual entry would use, with the same validation. Imported trunks come in disabled and imported outbound routes have no trunk attached, so nothing can route a real call until you review and enable it.

`Utilities → FreePBX import`

API & automation

Everything the console does is a REST call. This chapter is enough to build your own integration: how to get a key, how to authenticate, what to expect back, and where the full endpoint reference lives.

Open the interactive API reference (`/api-docs` on this server; it needs no key to read).

The basics

The API is the same one the admin console uses, not a separate product: every screen reads and writes through `https://<your-server>/api/...`. Requests and responses are JSON. There are two ways to authenticate: a **session** (what the console uses after you sign in) and an **API key** (what scripts and integrations should use). This chapter is about API keys.

> **API keys need a Pro license.** Creating, revoking and *using* a key all require a license that includes the `pro` feature (see Licensing). Without one, key requests are refused with HTTP 402. After a license expires, keys keep working for 14 days (responses carry an `X-RingStack-License: grace` header) before they stop. Signing in to the console is never affected.

1. Create a key

1. Sign in as an admin and open **Settings → API keys**.
2. Name the key after the integration that will use it (for example *crm-sync*). One key per integration lets you revoke one without breaking the others.
3. Choose a **role**, optionally an **expiration date**, and optionally limit it to specific **resources** (see below).
4. Copy the key when it's shown. **It appears once.** Only a hash is stored, so there is no way to view it again; if you lose it, revoke it and create a new one.

SETTING	WHAT IT DOES
`Role: full`	Admin-equivalent when the key has no resource scope: passes every "admin only" check, including managing users, security, firewall and backups. Use sparingly.
`Role: readonly`	Can only make `GET` requests. Any write is refused, whatever the endpoint. The right default for reporting and monitoring integrations.
`Resource scope`	Limits the key to the resources you tick (extensions, call logs, and so on). A key with no scope reaches everything its role allows. A scoped key gets 403 on everything else, and on the admin-only areas (users, security, firewall, backups, settings) even if its role is `full`.
`Expiration`	Optional date in the future. After it, the key stops authenticating.
`Revoke`	Takes effect immediately. The row stays in the list (marked revoked) as an audit trail.

The key list shows each key's first characters (its prefix) and when it was last used, so you can spot unused keys and clean them up. Keys look like `sb_…`; on a multi-tenant install a tenant's key carries the tenant name, `sb_<tenant>.…`, and only reaches that tenant's data. Box-level settings shared by every tenant (license, updates, patches, TLS) are refused to tenant keys.

2. Authenticate

Send the key on every request, in either header:

```
Authorization: Bearer sb_xxxxxxxxxxxx
X-Api-Key: sb_xxxxxxxxxxxx
```

Always use HTTPS; a key is a password. No session, cookie or CSRF token is involved when you use a key. (The CSRF check only applies to browser sessions.)

```
# List extensions
curl -s https://pbx.example.com/api/extensions \
  -H "Authorization: Bearer $RS_KEY"
```

3. Make requests

Reads are `GET`, creates are `POST`, edits are `PUT`, and removals are `DELETE`. Send `Content-Type: application/json` with a JSON body on writes.

```
# Create an extension (the SIP password is returned once, in this response)
curl -s -X POST https://pbx.example.com/api/extensions \
  -H "Authorization: Bearer $RS_KEY" \
  -H "Content-Type: application/json" \
  -d '{"extNumber":"230","displayName":"Front Desk 2","email":"desk2@example.com"}'

# Last 50 answered calls from extension 105
curl -s "https://pbx.example.com/api/call-logs?limit=50&src=105&outcome=answered" \
  -H "Authorization: Bearer $RS_KEY"

# Click-to-call: ring extension 105, then dial the number and bridge the two
curl -s -X POST https://pbx.example.com/api/calls/originate \
  -H "Authorization: Bearer $RS_KEY" \
  -H "Content-Type: application/json" \
  -d '{"extension":"105","destination":"5551234567"}'
```

Writes go through the same validation and side effects as the console. Creating an extension, for example, also writes the phone config and reloads Asterisk, and a click-to-call goes through your normal outbound routes, call recording and concurrent-call limit.

4. Handle responses

Errors come back as `{ "error": "a readable message" }` with a real HTTP status.

STATUS	MEANING
`200` / `201`	Success. `201` is a create; the body has the new record.
`400`	A field failed validation. The message says which.
`401`	Missing, wrong, revoked or expired key.
`402`	The install's license doesn't include what the request needs: API access itself (the `pro` feature) or a gated feature such as AI. See Licensing.
`403`	The key's role or resource scope doesn't allow this (for example a `readonly` key attempting a write).
`404`	No such record.
`409`	Conflict, such as an extension number already in use.
`429`	The key is being rate limited (below).
`502`	The request was valid but an underlying system call (Asterisk, a trunk, a backup target) failed.

Rate limits

Each key is limited to **600 requests per minute** by default, so a looping script can't hammer the PBX. The limit is a token bucket that refills continuously, so short bursts are fine and the sustained rate is what's bounded. Admins can change or turn it off under **Settings → API keys → Rate limiting**. Signed-in console sessions aren't throttled.

When you exceed it you get `429` with a `Retry-After` header (seconds) and `{ "error": "...", "retryAfterSeconds": N }`. Wait that long, then retry. The AI assistant endpoints have their own separate per-hour limits (same settings card).

```
// Node 18+: retry politely on 429
async function rs(path, opts = {}) {
  for (let attempt = 0; attempt < 5; attempt++) {
    const r = await fetch('https://pbx.example.com' + path, {
      ...opts,
      headers: { Authorization: 'Bearer ' + process.env.RS_KEY,
                 'Content-Type': 'application/json', ...opts.headers },
    });
    if (r.status !== 429) {
      if (!r.ok) throw new Error((await r.json()).error || r.status);
      return r.json();
    }
    await new Promise(ok => setTimeout(ok, (Number(r.headers.get('Retry-After')) || 1) * 1000));
  }
  throw new Error('still rate limited');
}
```

What's available

The API has about 550 operations. The groups below map to the console's own areas; the console's menu names are a good guide to where something lives.

AREA	PATHS UNDER `/API/`
Extensions, phones, provisioning	`extensions`, `phones`, `phone-models`, `phone-activation`, `phone-firmware`, `provisioning-settings`
Trunks and numbers	`trunks`, `dids`, `outbound-routes`, `inbound-routes`
Call handling	`ivr-menus`, `queues`, `ring-groups`, `time-conditions`, `call-flows`, `flow-modules`, `conference-rooms`, `parking-lots`, `feature-codes`, `custom-feature-codes`, `webhook-actions`, `audio-files`
Live control and history	`calls` (originate), `call-logs`, `recordings`, `voicemail`, `messages` (SMS), `dashboard`, `search`
Reports	`reports`, `report-schedules`
Security	`firewall`, `security`, `security-events`, `fraud`, `geoip`, `apiban`, `sessions`, `api-keys`, `users`, `teams`
System	`settings`, `system`, `backups`, `backup-jobs`, `backup-destinations`, `patches`, `ha`, `license`, `network-settings`
Multi-tenant	`global-admin` (platform administrators only; see below)

Full per-endpoint detail is in the interactive reference. For each operation it shows the path and parameters, the request body fields, the access it needs, and example responses where they are documented:

- **`/api-docs`** on your own server: browsable Swagger UI where you can read every endpoint. It is public, so you can read it before you have a key.
- **`/api/openapi.json`**: the same reference as an OpenAPI file. Feed it to a client generator (openapi-generator, `openapi-typescript`, and similar) to get a typed client in your language.

Reading an endpoint

- **Requires**: each operation lists what it needs beyond signing in. *Permission* `extensions` means a console user needs that permission, and an API key must be unscoped or list it (a `readonly` key can only read). *Admin* means an admin session or an unscoped `full` key. *Box owner* is the same, but on multi-tenant installs only a global admin. *Licence feature* means the license must include it, or you get HTTP 402.
- **Request body**: hand-written entries show types, allowed values and which fields are required. Entries tagged `x-generated` list the field names the server reads, without types; omit what you don't need and check the response.
- **Machine-readable**: the same requirements appear as `x-requires` (for example `permission:extensions`, `admin`, `feature:pro`) in the OpenAPI file.

The spec is generated from the server's own route code, so it matches the version you're running.

Multi-tenant installs

A tenant's admin creates keys for their own tenant under the same **Settings → API keys**; those keys can't see any other tenant. Platform administrators can also create **global** keys on the Global Admin console. These start with `sbg_`, are accepted only on `/api/global-admin/*` routes, and are how you automate tenant provisioning and platform-wide tasks.

Good practice

- Use `readonly` unless the integration must write, and scope it to the resources it needs.
- Give every integration its own key and an expiration date you will remember to renew.
- Keep keys in a secrets manager or environment variable, never in source control or a browser page.
- Treat `429` as normal flow control: back off using `Retry-After`.
- Don't poll for call state in a tight loop. Query call logs or the dashboard at a sensible interval, or use a webhook for events in the other direction.
- If a key leaks, revoke it. It stops working at once.

Users, teams & access

Users

Who can sign in to this console, and what they're allowed to do. Every account sets up two-factor authentication — an authenticator app (TOTP), a passkey (WebAuthn), or an email code — the first time it signs in, even accounts created before 2FA existed; it can be reconfigured afterward per-account.

`Settings → Users`

Single sign-on

Let your team sign in with an identity provider you already use, instead of a separate RingStack password.

`Settings → SSO`

Feedback

Notes left through the feedback bubble — nothing here is acted on automatically; it's reviewed and prioritized before any work starts.

`Feedback`

Security & system administration

Security

Who's trying to get in, and who's been blocked — backed by real fail2ban jails watching SIP registration and admin-panel logins. Bans are real firewall rules, not just a log entry. A repeat offender who keeps coming back after a normal ban expires gets escalated to a much longer block automatically. A real, observe-only rate log also watches general API traffic patterns. A caller-ID blocklist rejects known spam numbers before they ring anyone, across every DID at once — not tied to one number.

`Security`

Firewall

The actual UFW rules currently enforcing on this box, plus trusted sources — known carrier trunks and paired systems get scoped firewall access automatically and are excluded from fail2ban's auto-ban. Blocks by country (GeoIP) too, with per-IP exceptions for a legitimate call landing from outside an allowed country. Port 80 stays reachable from blocked countries only for Let's Encrypt's certificate checks, which come from several countries; nginx refuses everything else there, including phone provisioning.

`Firewall`

Network / NAT

Whether this box sits on a public IP directly or behind NAT — gets SIP/RTP addressing right either way.

`Network / NAT`

Utilities

Named backup jobs, each with its own content scope (database, config, voicemail, recordings, hold music, uploaded audio, branding) and its own remote destinations — see Backup & restore. Also where recurring report emails, how long call recordings/CDR are kept before auto-purge (data retention), live SIP packet captures, and the FreePBX importer all live.

`Utilities`

Patches

Pending updates for Asterisk, PJSIP, and the rest of the real stack underneath — read directly from the OS's own package cache. A known-unpatched CVE affecting your exact OS+Asterisk version is checked automatically by `ringstack-cli update` and applied as an override if one exists.

The same page has a **RingStack itself** card for updating *this application* (pick a channel, **Check for updates**, then **Apply update**). Its **Maintenance** row is what to reach for afterwards: **Reload configuration** regenerates the phone-system config and live-reloads it into Asterisk (safe with calls up — do it if the update changed how routing or phone config is generated), and **Restart Asterisk** does a full restart for changes a live reload can't pick up. If calls are active it asks whether to wait up to 30 seconds for them to end or restart immediately and drop them; phones re-register on their own. On a multi-tenant system only a global admin can restart Asterisk, since it drops every tenant's calls.

`Patches`

Notifications

Email/webhook/SMS alerts for the things worth knowing about without watching the console — a trunk going down, a backup failing, a new security incident, a critical patch available, the concurrent-call limit being hit, or a daily voicemail digest — each turned on or off independently, with its own routing to which channel(s) actually carry it. With AI integration on, any alert can optionally be summarized in plain language before it's sent, and the email includes a link straight back to the page where you'd actually do something about it (Firewall for a ban, Trunks for a registration failure, and so on). You can also define your own alert in plain language — "let me know if any trunk has been down for more than 10 minutes" — checked on its own schedule using the same real system data the built-in alerts use, and only fires the first time it becomes true, not on every check.

`Settings → Notifications`

Settings

Company details, branding, email delivery, mobile push credentials, licensing, and everything else that isn't day-to-day call handling.

`Settings`

SSL/TLS certificate

One certificate for the console, phone provisioning and SIP over TLS: Let's Encrypt (HTTP-01, or DNS-01 through Cloudflare or Route 53), your own upload, or self-signed. A new certificate waits for you to click **Apply**. Let's Encrypt renewals are checked daily from 30 days before expiry, and a renewal that changes only the dates applies itself; anything else (new names, a chain that doesn't verify) waits on this page with the reason. Asterisk restarts on Apply only when an extension or trunk uses SIP over TLS. The page also shows what port 443 actually serves, the last renewal check and when renewal starts, and you get an email 21, 7 and 3 days before expiry and when a renewal fails or waits for Apply. On a multi-tenant box the certificate belongs to the box, so only a global admin manages it. On an HA pair, the active node renews and the standby keeps a copy, so renewals carry on after a failover. HTTP-01 renewals need port 80 open to the internet; GeoIP country blocking leaves it open for Let's Encrypt's checks on its own. Certificates can use an RSA or an ECDSA (P-256) key; an uploaded certificate can be RSA or EC on P-256 or P-384. If your service provider manages the DNS-01 credentials from RingStack Central, the page says **Managed by** them and the credential fields are locked; **Release management** gives them back to you and keeps them working.

`Settings → SSL/TLS`

Email delivery & support access relays

Three genuinely separate things in RingStack are sometimes all called "the relay" — here's each one, and how to set it up.

SMTP relay (email delivery)

This is the outgoing mail server RingStack uses to actually send email — voicemail-to-email, scheduled report emails, missed-call notifications, invite/password-reset links, and system alerts (trunk down, security incidents) all go through it. There is no hosted relay built into the product — you point RingStack at a real SMTP server you control or subscribe to (a provider like SES, SendGrid, Postmark, your own mail server, etc.).

Configure it under **Settings → Email delivery** (host, port, encryption, credentials, from-address). On a multi-tenant install, a box-wide default relay every tenant falls back to unless they set up their own can be configured once via the CLI instead of per tenant:

```
ringstack-cli set-global-email-settings --smtp-host smtp.example.com \
  --smtp-port 587 --smtp-encryption starttls \
  --smtp-user your-user --smtp-password 'your-password' \
  --from-address noreply@yourcompany.com
```

Managed by your service provider. If an MSP manages this system from RingStack Central, they can set the SMTP relay for you. Settings → Email delivery (and, on a multi-tenant box, the global admin's SMTP relay card) then shows **Managed by <your provider>**: the server, user name and password are theirs, and the from address stays yours to change. To take over, click **Release management**. The settings stay in place and keep working, and your provider is notified that it was released. Before Central changes a multi-tenant box's shared relay, it sends a real test email through the new settings, so a bad password never stops every tenant's mail.

AI and voice keys from your provider. An MSP can also push the AI provider key (Anthropic, OpenAI or Google) and the ElevenLabs key from RingStack Central. Settings → AI integration then shows **Managed by <your provider>** on that card: the key is theirs until you choose **Release management** (recorded in the audit log), while the feature switches, the model, the voice and voicemail transcription stay yours to change.

A backup destination from your provider. An MSP can give each system an off-site backup destination from RingStack Central: an S3-compatible bucket, an SFTP server or an FTP server, with its own folder per system so systems never overwrite each other. It appears under Utilities → Backups → Destinations as **Managed by <your provider>** and is added to your daily backup job, which keeps its retention. A managed destination is read-only here: you can browse and test it, but not change or delete it. To take it over, open it and choose **Release management** (recorded in the audit log). It keeps working with the same settings, and your provider no longer updates it.

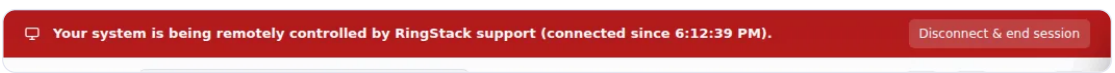
Support access relay

A self-hostable reverse tunnel that lets RingStack's own support team (or, if you run this yourself for your own customers, your own support team) reach a box for a real, time-limited shell session — without opening any inbound firewall port. Starting a session under **Settings → Support access** (or `ringstack-cli support-access-enable`) opens an outbound-only connection and grants access only to the trusted support key, only until the session ends or expires. Registering with a relay host is separate from starting a session: without a registered relay, a session can still grant the key locally, but there's no tunnel for a remote party to actually reach the box through. If you're white-labeling RingStack for your own customers, point the relay host field at your own infrastructure instead of RingStack's — it doesn't have to be RingStack's relay.

You approve every connection. By default a support session grants nothing on its own. When the technician connects, an amber banner appears on your console with an **Allow** and a **Deny** button (or run `ringstack-cli support-access-approve` / `support-access-decline`); only after you click Allow is the technician's one-time key installed. While they are logged in, a red banner reads “Your system is being remotely controlled by RingStack support”, with a **Disconnect & end session** button that drops them immediately. Connect and disconnect times are recorded in the session log. Uncheck **Require my approval** when starting a session (or pass `--no-approval` to `support-access-enable`) only for unattended work.



When a technician asks to connect, this banner appears at the top of every page. Only click Allow if you asked for support.



While they are logged in the banner turns red. Disconnect & end session drops them immediately.

Mobile push credential (not a relay)

Under **Settings → Mobile push**, the Team ID/Key ID/Bundle ID/Auth Key fields are your own real Apple Push Notification credentials (from your own Apple Developer Program account) — used directly, server-to-server, to wake the mobile app for an incoming call. This is a credential you supply, not a relay or hosted service RingStack operates on your behalf.

Licensing

A signed license file unlocks specific features on top of a genuinely free, uncapped core.

With no license applied, RingStack runs on its free default tier — every core telephony feature works, with no license required at all. A license file, once applied, unlocks specific additional features on top of that — it doesn't change your call/extension capacity, which is never capped by licensing.

Applying a license, whichever way you do it, is the same either way: a signed file goes to `/etc/ringstack/generated/license.json` and is re-verified from disk on every check (a JSON file read plus one signature check — cheap enough that nothing is ever cached). An invalid, tampered, expired, or missing license file always falls back cleanly to the free tier — it never fails in a way that breaks the system, and it never depends on reaching the network to keep working.

HOW	WHEN TO USE IT
`Settings → Licensing`	The normal path on any install that already has a reachable admin console — paste the license file content or the compact one-line code your vendor sent you.
`ringstack-cli apply-license /path/to/license.json` `ringstack-cli apply-license <the-license-code>`	No web session needed — works from a file or pasted content directly. On a **fresh** multi-tenant install this is the *only* way to unlock the `multitenant` feature, since `create-tenant` refuses to run until the box is licensed for it, and there's no reachable console yet before a tenant exists.
`ringstack-cli license-status`	Print the current license status (business name, feature list, expiry) as JSON — useful for scripting or a quick check.

A license is tied to the specific machine it was issued for (a hashed fingerprint derived from `/etc/machine-id` and the box's own MAC address, shown on Settings → Licensing) — installing it on a different machine falls back to the free tier with a clear "issued for a different machine" status rather than a silent failure.

Features actually gated behind a license today, verified against the current code:

FEATURE	WHAT IT UNLOCKS
`multitenant`	Creating a tenant on a multi-tenant install (`ringstack-cli create-tenant` / the Global Admin console) — required alongside the `--multi-tenant` install flag.
`ha`	High availability / active-standby clustering.
`ai`	Troubleshoot, report/call-journey/fraud-alert narration, the Pro Reports Ask assistant, recording transcription, and AI-generated audio-library voice prompts. (The AI provider key itself is a separate, bring-your-own credential under Settings → AI integration — the license controls whether these features are reachable at all, not which model they use.)
`branding`	**Branding** — your logo (with an optional dark-mode logo), accent color, favicon and phone-display logos on the console, sign-in page, generated PDF reports and branded emails, shown with a small “Powered by RingStack”. The product is still called RingStack. Uploaded SVGs are cleaned of anything that could run code before they're saved.
`whitelabel`	**White label** — everything in Branding, plus your own product name wherever the console, sign-in page, emails and PDFs would say “RingStack” (leave it blank to use the system name), and no “Powered by RingStack”. A branded mobile app isn't available yet.
`pro`	Creating and using API keys (a 14-day grace period applies after expiry), and Paired systems (create/edit/delete).
`central-mgmt`	Connecting this install to a RingStack Central fleet-management console (see RingStack Central).

Whatever isn't listed above — extensions, trunks, routing, queues, ring groups, voicemail, SMS, backups, security hardening, and everything else described in this document — is free on the core tier, uncapped.

Multi-tenant & the Global Admin console

Running several separate companies on one install.

A multi-tenant install keeps one shared control-plane database (the tenant registry, global admin accounts, box-wide defaults like the SMTP relay) plus a completely separate database per tenant — each tenant's extensions, trunks, routes, CDR, and everything else is fully isolated from every other tenant's. Requires a fresh install started with `--multi-tenant` (see Installing & getting started) and a license with the `multitenant` feature — an existing classic install can't convert in place.

Managing tenants is a separate login realm from any one tenant's own admin console — a *Global Admin* account (created during `--bootstrap` install, or via `ringstack-cli create-global-admin`) signs in at the same login screen and lands in its own Global Admin console instead of a tenant's regular admin shell. From there — or from `ringstack-cli`, which is often faster for scripting — you can:

- Create a new tenant, optionally cloning an existing tenant's full configuration as a starting point, or restoring one from an existing classic install's own backup archive (every real user carries over, signing in immediately with their existing password).
- List every tenant and its status.
- Permanently delete a tenant — database, media files, real-time registration state, and control-plane records, with the shared dialplan/PJSIP config regenerated immediately so its endpoints and routes disappear right away.

See the CLI reference for the exact `create-tenant` / `list-tenants` / `delete-tenant` commands and flags. Once a tenant exists, that tenant's own admin (or the Global Admin, "swapped in" to it) signs into and uses that tenant's console exactly like a classic single-tenant install — everything else in this document applies unchanged, per tenant.

RingStack Central (fleet management)

One console for watching, patching and billing many separate RingStack installs — built for MSPs.

Central is a **separate application** from the console you're reading about: it has its own login, its own database, and its own web address, and it runs as its own Docker stack (Central plus MariaDB). It doesn't replace each customer's own console — it sits above them. Each customer install ("site") stays a normal RingStack box; Central connects to it over that site's HTTPS API using an API key generated on the site, and never needs shell access to it.

Because **Central calls each site** (not the other way round), the Central host must be able to reach every site's URL over HTTPS — same network, a VPN, or a publicly reachable address. A site behind NAT with no route back to Central shows up as unreachable.

Getting Central running

Central ships as a Docker Compose stack in the `central/` folder: Central, MariaDB, Caddy (the only part that listens, on ports 80 and 443) and a backup helper. Install it from the **signed bundle** Corner IT sends you (`sudo sh verify-release.sh --keys release-keys --install-dir /opt/ringstack/central central-vX.Y.Z.bundle.tar`, then `sudo sh scripts/upgrade.sh --bundle …`; no registry, source or internet needed), or from a release tag. `./init-env.sh` writes the `.env` with fresh secrets and a first admin login, and **asks for Central's hostname** (or pass `--domain central.example.com`): that becomes the name Caddy serves and the address your systems are given. Choose how it gets its certificate with `--tls acme` (Let's Encrypt, a public name), `--tls own` (a certificate and key from your own CA in `central/certs/`, for a private network; check them with `sudo sh scripts/check-tls.sh`) or `--tls internal` (Caddy's own CA, for labs; then `sudo sh scripts/export-ca.sh`). To change the hostname later run `sudo sh scripts/set-domain.sh central.new-name.example`; enrollment codes and seeds made before then carry the old address and must be re-made.

The complete walk-through, with every screen and command, is the **Central administrator guide** (`docs/central-admin-guide.md`; with Central running it opens from **Settings → About**). `central/DOCKER.md` is the short deployment reference.

> **Back up `RS_ENCRYPTION_KEY` right away.** Central stores every site's API key encrypted with it (in `central/.env`). It isn't part of a database backup, and if it's lost every site's stored key is unrecoverable — each site would have to be re-added. Keep an offline copy.

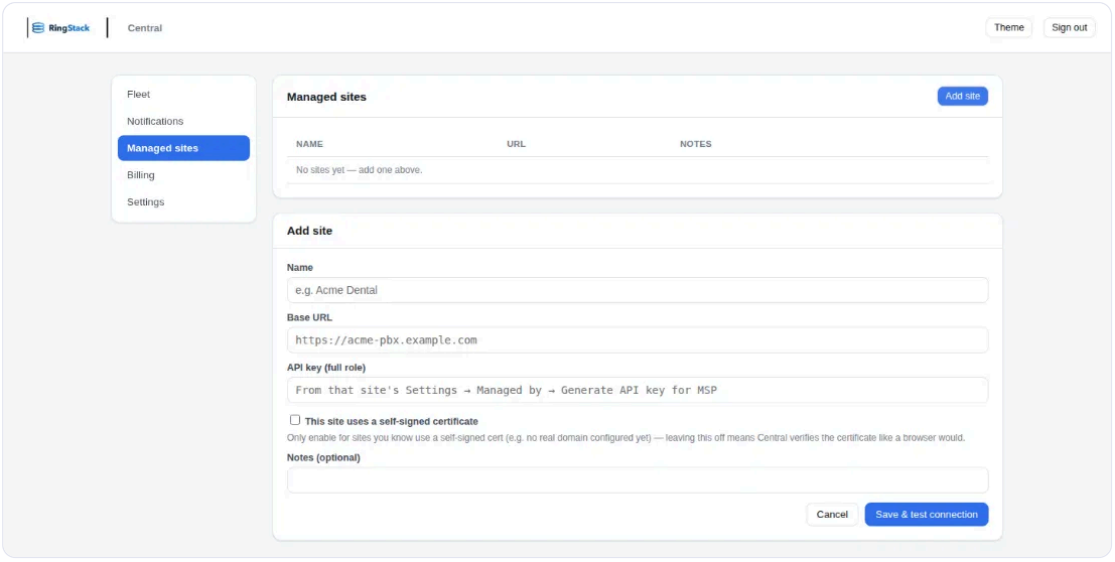
First sign-in. Sign in with the admin email in `.env` and the password `init-env.sh` printed. Every Central account must set up two-factor sign-in (an authenticator app) at its first sign-in and gets ten one-time backup codes; if you lose the authenticator an operator resets it on the host (`docker compose exec central node src/lib/cli.js reset-2fa you@example.com`). Then apply **Central's own license** under **Settings → License**: send the fingerprint shown there to your provider, paste back the license and choose **Apply license**. Until it is applied Central is in setup mode: you can sign in and read everything, but it will not enroll or change systems. Set up **Settings → Backups** before you add real systems (below).

Adding a system

The normal way is an **enrollment code**, so nobody copies a key by hand. In Central open **Managed sites → Add system**, add a label and choose **Create code**; copy the code now (it is shown once, works once, and lasts 1 hour to 7 days). On the system either open **Settings → Managed by → Connect to Central** and paste it, or run `sudo ringstack-cli enroll '<code>' --url https://this-system.example.com` (a new install: `sudo ./install.sh --enroll '<code>' --system-url https://…`). The system makes its own management key and sends it to Central, which checks it before listing the system. For a whole fleet use **Many codes…** and `sudo sh scripts/enroll-fleet.sh --codes codes.csv --hosts hosts.txt`, which enrolls existing systems over SSH in waves.

The older way, with a pasted key.

1. **On the site:** sign in as an admin and open **Settings → Managed by**, then choose **Generate API key for MSP**. Copy the key when it's shown — it appears once. Generating a new one revokes the previous MSP key. This button requires a license with the `central-mgmt` feature on that site (see Licensing). The key has the full role.
2. **In Central:** open **Managed sites → Add site** and fill in **Name**, **Base URL** (e.g. `https://acme-pbx.example.com`, no path), **API key**, and optionally **Notes**. Tick **This site uses a self-signed certificate** only for a site you know has one; otherwise Central verifies the certificate the way a browser would.
3. Choose **Save & test connection**. Central calls the site before saving; if it can't connect it tells you why and doesn't save.



Managed sites → Add site on a fresh Central with no sites registered yet.

Onboarding a new customer. In Central, open **Customers → New customer** and enter the customer's name and the email of their first admin. Then connect their system: a **new install** or an **existing system** gets a one-time command with an enrollment code (it works once, for 24 hours, and is shown only once); a **tenant on your multi-tenant box** is created for you, invite-only and optionally copied from one of your starter templates, and connects itself. For tenants, connect the box itself with **Let Central add a tenant for each new customer** turned on (its global-admin Central card, or `ringstack-cli enroll … --box --allow-tenants`). The customer's checklist then walks through the license, pushing credentials and branding, and inviting the first admin. The customer goes **live** once their admin has signed in, set up two-factor sign-in and accepted the license agreement; billing starts on the day their system connects. Scripts can do the same through Central's automation API with a key from **Settings → Automation API keys** (keys never reach the credential vault).

License agreement. When a license agreement is published on a system, an admin is asked to accept it once, after setting up two-factor sign-in, on behalf of their organisation; a new version asks once more. Only the customer's own admin can accept: your MSP, RingStack Central and API keys can see that it was accepted (who, when, from where) but cannot accept for you.

A system without a Central license yet. It can still take the code: it then **waits for its license from the MSP** instead of connecting. Central lists it under **Managed sites → Enrollment codes** as *Waiting for license*, with its hostname and machine fingerprint; choose **Attach license** and give it the license issued for that fingerprint (or apply a license bundle — waiting systems are matched too). Within 5 minutes the system applies the license and finishes connecting by itself. The wait lasts 14 days, and the system's **Managed by** card shows it with a **Stop waiting** button. On a multi-tenant box, connect the box itself first; its tenants connect once it is licensed.

Many systems at once. Managed sites → Many codes… makes up to 200 codes from a list of labels and gives you a CSV with each code and its installer line (`sudo ./install.sh --enroll '<code>' --system-url https://…`), so a new system connects as part of its install. A code works once, and a copied disk image can't register twice: the same machine at a second address is refused until it is decloned.

Licenses. The **Licenses** tab shows every system's license — who it's issued to, its features, when it expires, whether it includes Central management and whether API access is in its grace period — read every 6 hours, with alerts at 30, 14, 7 and 1 days before expiry. **Apply license…** pushes a license to one system (it needs a fresh authenticator code); **Apply license bundle** previews a bundle file matched by machine fingerprint before anything is sent. A system that refuses a license keeps the one it has. To get many licenses at once, **Export waiting systems** downloads a CSV of the machine fingerprints that are waiting; send it to your provider, who returns one bundle file for **Apply license bundle**.

The site's own **Managed by** card shows whether a live MSP key exists and when it was last used. To rotate a key, generate a new one on the site and paste it into the site's entry in Central.

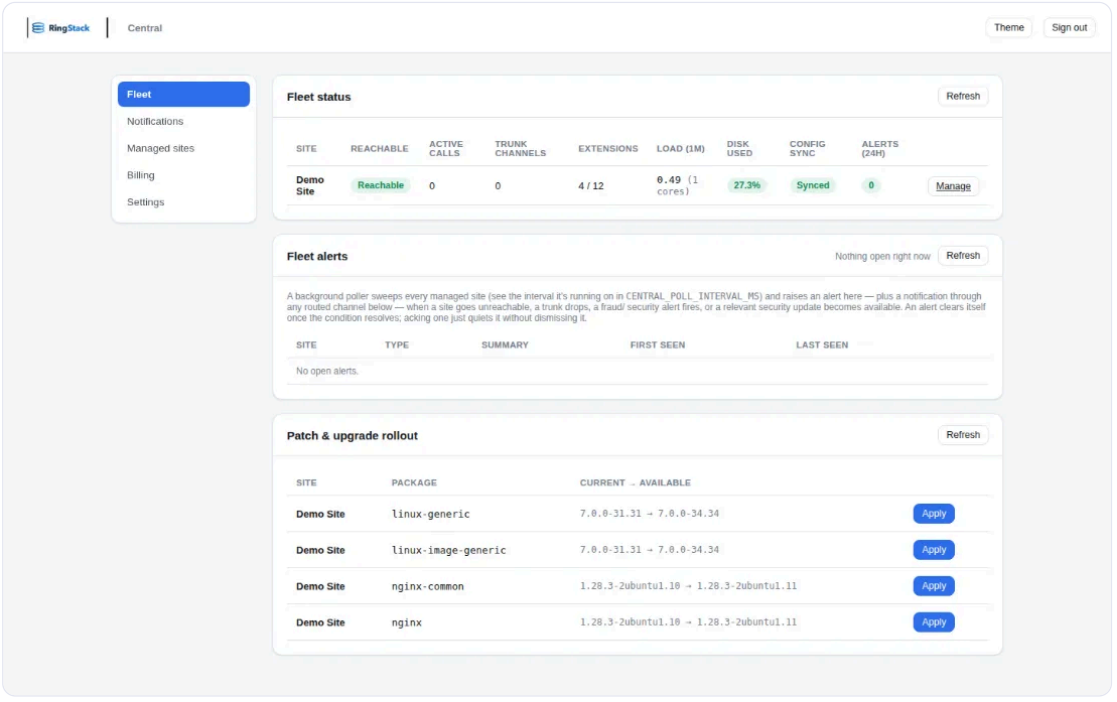
Multi-tenant boxes. Each tenant connects as its own site. The box itself can connect too, so Central can see its license, certificate, updates and health: in the global-admin console open **License → RingStack Central**, paste a code from **Add system** and choose **Connect box** (or run `ringstack-cli enroll <code> --url https://box.example.com --box`). In Central the box appears with a **Box** badge, with its tenants listed under it. **Disconnect** revokes the box's key without touching the tenants.

Fleet, alerts and notifications

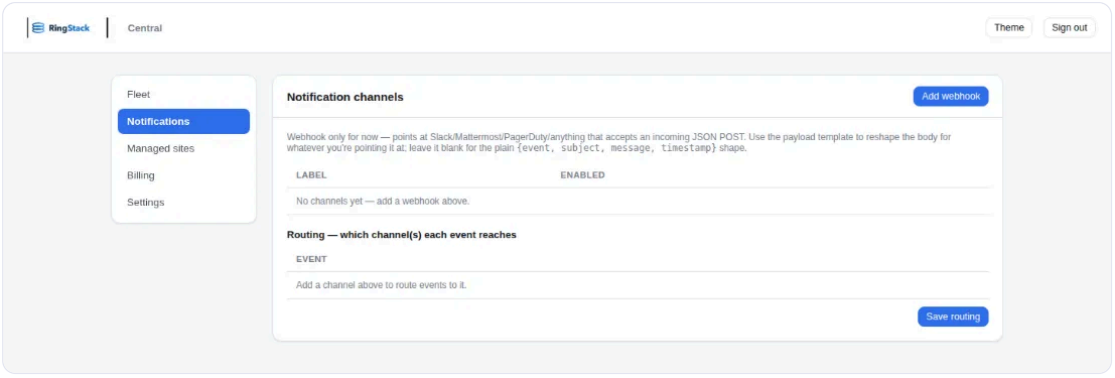
- **Fleet** tab — **Fleet status** lists every site with whether it's reachable, active calls, trunk channels, extension count, 1-minute load, disk used, config-sync state and alerts in the last 24 hours. Each row has a **Manage**

button that opens that site's full RingStack console in a new tab, served through Central (Central adds the site's key on its side — your browser never sees it).

- **Fleet alerts** — a background poller sweeps every site (once a minute by default) and raises an alert when a site becomes unreachable, a trunk drops, a fraud/security alert fires on a site, or a relevant security update becomes available. An alert clears by itself once the condition resolves. Acknowledging one quiets it without dismissing it.
- **Notifications** tab — **Add channel**. An **E-mail** channel sends to a list of recipients through Central's own SMTP (**Settings → Central's own SMTP**; use its **Send test email** first). A **Webhook** channel sends alerts to Slack, Mattermost, PagerDuty or anything that accepts a JSON POST, with a **Webhook URL**, an optional **Shared secret** (sent as an `X-Central-Webhook-Secret` header) and an optional **Payload template**; leave the template blank for the plain `{event, subject, message, timestamp}` shape. A receiver on your own network is refused unless its range is listed in `CENTRAL_WEBHOOK_ALLOW_CIDRS` in `.env`. Each channel shows whether its last delivery worked and has **Send test**. Under **Routing** choose which channel(s) each event reaches (fleet, credentials, certificates, licenses, customers, billing, updates, Central backups and provisioning events), then **Save routing**. There is no SMS channel.
- **RingStack versions** — one row per system and per multi-tenant box (its tenants update with it): the installed commit and channel, whether an update is available, how many releases behind it is, the last update and who started it, the box's update window in its own time zone, and any pin. An HA standby node is updated with its active node. **Policy** stores a box's window and pin on the box itself: inside its window the box installs updates on its own, even while Central is down, and never past a pinned commit.
- **Rollouts** — select systems and choose **Roll out to selected** (or **Update** on one row). The canary (one box by default) updates first and nothing else starts until it has; the commit it installs is the one every other box must install. Then waves of 5 (smaller if you like, never off), with 30 minutes per box. The first failure stops the rollout: boxes already updating finish, nothing new starts, and you can **Resume**, **Resume and retry failed** or **Abort**. Boxes on another channel, HA standby nodes, pinned boxes, boxes already up to date and unreachable boxes are skipped and listed with the reason. A rollout survives a Central restart. Starting or resuming one asks for a fresh authenticator code; stopping never does. Alerts: a failed update, and a site more than 2 releases behind.
- **OS patches** — lists, per site, packages with an update available (current → available). **Apply** installs one package on one site after a confirmation; this may restart services on that site. **Apply all** applies each package on every site that needs it, at most 5 sites at once. Each apply waits for the site's result, and reports failure or a timeout rather than assuming success.



The **Fleet** tab: **Fleet status**, **Fleet alerts**, **RingStack versions**, **Rollouts** and **OS patches** with one site registered.



The **Notifications** tab before any webhook is added.

Credentials (the vault)

The **Credentials** tab stores a third-party secret once and puts it on every system that should use it: AI keys, Email (SMTP), Voice (ElevenLabs), GeoIP (MaxMind), APIBAN, Apple push, Backup storage, Trunk passwords and Certificates (DNS) credentials. It needs an admin with the **Manage credentials** permission and a fresh authenticator code for each change. Secrets are write-only: Central shows only a short fingerprint. In **Site assignments** each system is **Shared** (a stored key), **Dedicated** (its own secret) or **Unmanaged**; **Apply** merges the value in, marks the setting as managed (read-only on the system) and reads it back. **Rotate** tries the new secret on a canary system first and rolls it out to the rest only if that works. A setting changed on the system shows **Drifted** and raises an alert; Central never overwrites on its own. **Revoke everywhere** clears a leaked secret from every system. The Credentials tab also holds **Branding for your sites**: make a brand (logos, favicon, phone logos, accent colour, optional product name), make it the MSP brand and **Push to all sites** after Central shows how many systems will change; each system can follow the MSP brand, have its own brand, or be left alone. The product name replaces “RingStack” only on systems with the White label license; the Branding license pushes images and colour.

Certificates

The **Certificates** tab lists every system's TLS certificate, what it serves, when it expires, its source and whether it renews itself, and alerts at 21, 7 and 3 days. Admins with **Manage certificates** can use **Renew now**, **Apply new certificate**, **Turn auto-renew on / off**, **Change hostname…** (Central checks the DNS first, gets the new certificate and applies it only when it is ready) and **Upload a certificate…** for a customer's own certificate, optionally keeping an encrypted copy to redeploy. **Renew expiring** renews every Let's Encrypt certificate that is within 14 days of expiring.

Provisioning (Proxmox templates and clones)

To stand up many systems from one VM template: prepare the template with `ringstack-cli seal-template` (it removes every secret and identity), then in **Provisioning** choose **New deployment** (how many hosts, names, DHCP or static addresses, approval rule, a **Config profile**). For each host **Make seed** gives the cloud-init files and the `qm set` command; clone and start the VM in Proxmox (Central never talks to Proxmox). Each clone resets its identity, checks in, and waits for your **Approve**. A clone with no seed appears under **Systems waiting without a seed** and holds nothing until you approve it into a deployment. Then **Export waiting fingerprints**, have licenses issued, import the bundle, and each host is configured and marked ready. Scripts do the same through the `/api/v1/provisioning` API with a key from **Settings → Automation API keys**.

Billing

Central works out what each of your customers owes under rate plans you define, and produces statements you can hand to your own invoicing tool. **Central never takes payment.** Every part of the Billing tab is described below.

CARD	WHAT IT DOES
`Rate plans`	A plan has a **currency** (USD, CAD, EUR, GBP), a **flat base fee**, and **extension tiers**. One tier with no upper bound is a flat per-extension rate; several tiers are either **Graduated** (each rate applies only to the extensions within its tier) or **Volume** (crossing a threshold reprices every extension). Optionally add **call minutes** (a flat rate with minutes included, or a rate-deck markup) and **storage** (GB included, rate per extra GB). Leave minutes or storage at zero and no line item is added.
`Rate decks`	Upload a carrier's per-minute cost table as a CSV, mapping your file's own column headers to dial prefix, cost per minute, and optionally description, connect fee and billing increment. A plan set to "rate-deck markup" then rates each call against the deck and adds your percentage or flat per-minute markup.
`Simulator`	Pick a plan, enter a hypothetical extension count, call minutes and storage, and choose **Calculate** to preview the total. Nothing is saved.

CARD	WHAT IT DOES
`Billing clients`	A billing client is who gets billed: a name, a rate plan, an optional billing email, and one or more managed sites (a multi-location customer's sites appear on one consolidated statement).
`Statements`	**Generate** a statement for a client and period. It uses the usage figures Central has already recorded for that client's sites, so a brand-new site has to be polled first ("no usage data yet" otherwise). Each statement offers **PDF**, **CSV** and **Import CSV** downloads, and a total is shown across clients converted to a currency you pick.
`Statement schedule`	**New schedule** to generate and email statements automatically — for one client or all clients, weekly or monthly (with a day of the month), to a comma-separated list of recipients. Schedules are checked hourly, and email needs Central's own SMTP (below).

Usage is recorded from each site once a day by default: the extension count, call minutes for the period, and stored recording/voicemail bytes. Statements are only as fresh as the latest recorded snapshot.

Settings, team and branding

- **Central's own SMTP** — host, port, username, password and From address. Central has no email capability of its own without it, and it is separate from any site's mail settings. Scheduled statements and E-mail notification channels use it.
- **Branding** — replace the logo (light and dark backgrounds) and favicon shown on Central; anything you don't upload keeps the default. **Reset to default** reverts one.
- **Branding for your sites** (Credentials tab) — make a brand once (logo and dark-mode logo, favicon, phone logos, accent colour, and the product name shown in place of “RingStack”), make it the MSP brand, and push it. Before anything is sent, Central shows how many sites change; **Revert to previous brand** puts the last MSP brand back. In **Site branding** each site follows the MSP brand, gets its own brand, or is not managed. Sites need the Branding license (otherwise they show **Branding not licensed**); demo tenants are never changed. If someone changes a site's branding on the site itself, Central shows **Changed on the site** and leaves it alone until you push again or release it.
- **Backups** (**Settings → Backups**) — Central's database and branding, encrypted with an `age` public key you paste in, to S3-compatible storage, SFTP, FTP or a local folder, on daily or weekly schedules with retention, a **Test** for each destination, **Pull back to this host**, and alerts when a backup fails or none has worked for 26 hours. Keep the `age` private key and `RS_ENCRYPTION_KEY` offline in two places, and confirm both under **Backup escrow and restore**. Restore is a host command (`sudo sh scripts/restore.sh <backup> <key>`); rehearse it before you rely on it.
- **Automation API keys** — keys for scripts that onboard customers and provision hosts through `/api/v1`, each with named scopes, optional allowed addresses and an expiry. Shown once.
- **Support bundle** and **About** — **Download support bundle** collects Central's state with secrets removed (read it before sending); **About** shows the version, commit, release tag, license, public address, uptime, database version and last backup, and opens the administrator guide.
- **Team** — **New user** creates a sign-in with a role (every account uses two-factor sign-in). An **Admin** sees and does everything. A **Client-scoped** user sees only the Billing tab and only the billing clients you assign them, which suits handing a customer their own statements. Passwords must be at least 12 characters; sessions end after 30 minutes idle or 12 hours. Admins can also be given **Manage credentials** and **Manage certificates** separately.

Troubleshooting Central

Adding a site fails, or the site shows as unreachable.

Central couldn't connect from its own host. From the Central server, run `curl -I <base-url>` — if that fails, it's a routing or firewall problem (NAT, VPN, DNS) between Central and the site, not a Central setting. Check the URL has no trailing path.

The error mentions a certificate.

The site's certificate isn't trusted by Central (self-signed, or an incomplete chain). Fix the site's certificate, or, for a lab site you know is self-signed, tick **This site uses a self-signed certificate**.

The site rejects the key (401/403 style error).

Regenerate it in **Settings → Managed by** on the site and paste the new one into the site in Central. Generating a new key revokes the old one, so a stale key stops working. Note the site needs a `central-mgmt` license to generate it.

Sites that were working start failing after Central was rebuilt or restored: `RS_ENCRYPTION_KEY` changed, so the stored keys can't be decrypted — restore the original key, or re-add each site.

Generating a statement says there's no usage data.

Central records usage for each site on a schedule (daily by default), so a newly added site has none until its first snapshot is taken.

Everything is read-only, or enrolling is refused with a license message.

Central's own license is missing or has lapsed past its grace period: open **Settings → License**. Calls and phones on your systems are never affected.

Locked out, or lost the authenticator.

On the Central host run `docker compose exec central node src/lib/cli.js reset-2fa you@example.com` (or `reset-password`, `unlock-login`). The administrator guide lists the rest, including upgrades (`scripts/upgrade.sh`, `scripts/rollback.sh`) and support bundles.

API reference for Central itself is in `central/API.md`; the full administrator guide is `docs/central-admin-guide.md`.

High availability

Active/standby clustering across two RingStack nodes.

Two nodes share a floating IP; the active node serves traffic, and the standby takes over automatically if the active node goes down. Configuration, registrations, and the database all recover on the standby — a call already in progress at the exact moment of failover drops (mid-call state isn't preserved), but everything else continues normally. Requires a license with the `ha` feature on each node.

Set up from **Settings → High availability**, or via `ringstack-cli`:

STEP	COMMAND
1. Enable HA on the first node	`ringstack-cli ha-enable-solo <vip-address>`
2. Generate a join token on that node	`ringstack-cli ha-join-token`
3. Join the second node using that token	`ringstack-cli ha-join <token>`

After that, failover is automatic (driven by `keepalived`) — `ha-promote` / `ha-demote` fire on their own when a node's role changes, and `ha-health` is the underlying reachability check keepalived polls. `ha-sync-ip` / `ha-peer-sync-ip` show or set which local address each node uses for replication/file-sync traffic if the default candidate isn't the right one on your network.

Backup & restore

Named backup jobs for day-to-day protection; whole-box archives for moving hardware.

Backup Jobs (Utilities → Backups) are the normal, ongoing way to protect one install (or one tenant): each job has its own content scope — database, config, voicemail, recordings, hold music, uploaded audio, branding — and its own remote destination(s) and schedule. Run one on demand with `ringstack-cli backup --job <slug>`, or let the scheduled-backup timer fire them automatically.

Whole-box backup (multi-tenant installs only) captures everything in one archive — the control-plane database, every active tenant's own database, and every tenant's media — for moving an entire box (an OS upgrade, new hardware), not as a replacement for per-tenant Backup Jobs:

```
ringstack-cli backup-box
ringstack-cli list-box-backups
ringstack-cli restore-box /path/to/archive.tar.gz --confirm
```

`restore-box` refuses to run unless the target box has zero tenants already — it's meant for restoring onto a fresh install, not merging into an existing one.

Every backup this version produces bundles its own encryption key inside the archive, so restoring it (via `create-tenant --from-backup` or `restore-box`) automatically decrypts and re-encrypts every SIP/trunk/webhook/AI-key secret with the new box's own key — nothing manual needed unless the archive predates that (pass `--source-encryption-key` to override or supply the old box's key by hand in that case).

ringstack-cli reference

Admin/ops utility, no web session needed — `ringstack-cli <command> [--flag value ...]`. Run `ringstack-cli help` any time for this same list from the box itself. On a multi-tenant install, a command that operates on one tenant's own data needs `--tenant <slug>`; commands that operate on the shared control plane (tenant/global-admin registries, licensing, HA, box-wide email defaults) never take one.

Tenants & global admins *(multi-tenant only)*

`create-tenant --slug S --name N --admin-email E [--admin-password P] [--clone-from SLUG]`	Create a tenant: real database, every migration applied, first admin, ready to sign in immediately. `--clone-from` copies an existing tenant's full configuration (and media) as the starting point instead of building an empty schema.
`create-tenant --slug S --name N --from-backup /path/to/archive.tar.gz [--source-encryption-key K]`	Create a tenant by restoring an existing classic install's own backup — every real user comes along, signs in immediately with their existing password.
`delete-tenant --slug S --confirm`	Permanently delete a tenant. Omit `--confirm` to preview what would be deleted first.
`list-tenants`	List every tenant and its status.
`create-global-admin --email E [--password P] [--name N]`	Create a Global Admin account — a separate login realm from any tenant's own users.

`reset-global-admin-password --email E [--password P]`	Break-glass password reset for a Global Admin — forces a change on next login.

Licensing

`apply-license /path/to/license.json | <license-code>`	Apply a license — a real file, or the compact single-line code your vendor sent. No web session needed.
`license-status`	Print the current license status as JSON.

First-run & account setup

`create-admin [--email E] [--password P] [--name N]`

`set-global-email-settings --smtp-host H [--smtp-port P] [--smtp-encryption starttls|tls|none] [--smtp-user U] [--smtp-password`

`seed-demo [--email E] [--password P] [--base-url U]`

System lifecycle & maintenance

`reload`	Regenerate every config file from the database and reload the affected Asterisk subsystems — run this after a change made outside the normal UI flow, or if something looks stale.
`asterisk-reload`	Raw "core reload" over AMI, no config regeneration — for when the on-disk config is already correct and Asterisk just needs to re-read it.
`asterisk-restart [--force]`	Restart Asterisk — waits up to 30s for active calls to drain first; `--force` restarts immediately, dropping any active calls.
`restart-all [--force] [--include-db]`	Restart every core service in one motion (Asterisk, the backend, nginx, tftpd-hpa, and keepalived if HA is enabled) — for when things go sideways and you want a known-good state without restarting each piece by hand. The database engine is skipped unless `--include-db` is passed (real risk: drops every live connection, and on a paired HA node can disrupt replication).
`status`	Database/AMI/service reachability plus extension/trunk/phone counts, in one glance.

`fix-permissions`	Re-apply correct ownership/mode to every install-managed path — run this if something looks like a permissions problem after manual intervention. On multi-tenant installs it also covers every tenant's audio, branding, recording and voicemail directories, and needs no `--tenant`.
`check-asterisk-security [--confirm]`	Check whether a known-unpatched CVE affects this box's exact OS+Asterisk version, and apply the override package if one exists and `--confirm` is passed. Also runs automatically as part of `update`.
`update [--check] [--confirm]`	Pull the latest release, apply pending migrations, redeploy, and restart. `--check` reports what would change without applying it; `--confirm` applies it.
`declone [--force]`	Regenerate every secret a cloned "golden image" VM would otherwise share with its template (SESSION_SECRET, DB/AMI passwords, SIP credentials, API keys, self-signed TLS cert) — your actual configured data (extensions, trunks, routes) is untouched. Fires automatically on first boot of a clone made from an `--is-hosted` image; `--force` runs it by hand.

Break-glass recovery

`reset-password --email E [--password P]`	Reset an admin's own password directly — generates a strong one if `--password` is omitted.
`unlock <code> [--email admin@example.com]`	Signs a specific already-open browser tab in as an admin, using a one-time code minted from that tab's own "stuck at login" screen — real console break-glass recovery, not a global bypass.
`decrypt-credential --type ext|trunk --id ID`	Recover a real SIP username/password for an extension or trunk — e.g. to hand it to an on-site technician without giving them an admin login. Logged to the audit log.

Dangerous — resets real data

`db-init [--confirm]`	Drop and recreate the entire database, then re-run every migration — first-install state. On-disk assets (TLS certs, audio library, logo, voicemail recordings) are **not** touched.
`factory-reset [--confirm]`	`db-init` plus wipe voicemail/recordings/audio/hold-music/branding — genuine first-install state end to end. Local backup archives and TLS certificates are deliberately left alone.

Backup & restore

See Backup & restore above for `backup`, `backup-box`, `list-box-backups`, and `restore-box`.

High availability

See High availability above for `ha-enable-solo`, `ha-join-token`, and `ha-join`. Also available: `ha-promote` / `ha-demote` (fired automatically by keepalived on a role change), `ha-health [--verbose]` (the reachability check keepalived polls), `ha-file-sync` (pulls changed files from the peer if this node is standby), and `ha-sync-ip` / `ha-peer-sync-ip [<ip>|clear]` (show or set which local address this node, or its peer, uses for HA traffic).

Support access

`support-access-approve / support-access-decline`	Allow or refuse the technician's waiting connection request. `support-access-status` shows when one is waiting and whether someone is connected now.
`support-access-enable [--ttl hours] [--no-approval]`	Open an ad hoc support session (default 8h) — grants the vendor support key and opens the relay tunnel if registered with one. See Email delivery & support access relays.
`support-access-disable`	End the active support session now, revoke the vendor key.
`support-access-status`	Whether a support session is active, and its expiry.
`support-access-add-key "ssh-ed25519 AAAA..." --label "Name"`	Trust another vendor support key — a real team, not one shared key. Takes effect on the next session started.
`support-access-remove-key ID`	Stop trusting a vendor support key.

`support-access-list-keys`	List every vendor support key ever added, active or revoked.

Timer-fired commands

These run automatically on their own schedules (systemd timers) — documented here for reference, or in case you need to run one by hand.

`run-scheduled-backups / run-scheduled-box-backups / run-scheduled-reports / run-queue-callbacks`	Run every backup job / whole-box backup / report email / pending queue callback whose own schedule is currently due.
`seed-moh-library`	Seed the bundled default hold-music groups (no-op if already seeded).
`sample-call-activity`	Record one active-call sample — feeds the Dashboard's peak/trend numbers.
`prune-continuous-pcap / purge-old-call-data`	Delete continuous packet-capture files, or CDR rows/recordings, past their configured retention.
`resolve-trust-zone / refresh-geoip`	Re-resolve trusted sources and reapply firewall rules; re-download stale GeoIP blocklist data.
`fix-geoip-port80`	Let GeoIP-blocked countries reach port 80 for Let's Encrypt only (nginx refuses the rest); run by update.
`set-trusted-proxies`	Front proxies (for example 127.0.0.1 for a Caddy on the same box) whose X-Forwarded-For nginx believes, so audit, rate limits, sign-in lockouts and fail2ban see the real client. No arguments shows the list; `--clear` empties it (the default). Run by install.sh from RS_TRUSTED_PROXIES; update re-applies it.
`run-cdr-relay / run-queuelog-relay`	*(multi-tenant)* Route the shared raw CDR/queue-log data into each tenant's own database.
`check-notifications`	Poll trunk/security/patch status and email on new events.

`tls-check`	Check the box's TLS certificate now: record renewals, check what port 443 serves, send due certificate alerts (also part of check-notifications).
`voicemail-digest`	Email a daily new-voicemail summary, if enabled.

Classic vs. Guided navigation

The left-hand navigation can be grouped two ways. **Classic** lists every page individually, grouped by category — best once you already know where things live. **Guided** highlights the wizard-driven flows (Set up a number, the Call flow map, Route Checker) more prominently for a less error-prone path through the same features. Switch between them anytime from the navigation itself; nothing about your configuration changes based on which mode you use — it's a display preference only.

Guided tour

A demo business starts a short guided tour the first time you sign in: the dashboard, one call's path and **Explain this call**, a question already typed into **Reports → Ask**, who was available to answer, voicemail transcripts, the call flow map, the browser phone, and what the demo can't do. Each step points at the real page. The tour never runs the AI for you: press **Ask** or **Explain this call** yourself.

Use **Next** and **Back** (or the arrow keys) to move, and **Skip tour** (or Esc) to close it. The **Tour** button at the top starts it again, or picks up where you left off. For now the tour runs only in demo businesses; a version for every install, which an admin can turn off, is planned.

Troubleshooting & FAQ

Symptom, real cause, real fix.

> **Before anything else:** turn on AI integration and use Troubleshoot — it traces the dialplan for a specific number, checks live trunk/extension registration against what's configured, reads recent Asterisk log errors, and can pull the exact SIP signaling for a failed call, all read-only. For a large share of "why didn't this call work" questions, it's faster than working through the FAQ below by hand.

A phone won't register / shows "Unregistered."

Check the extension's SIP credentials against what's actually configured on the phone (Extensions → the extension itself, or re-provision it from Phones). If the phone is off-site, confirm Network/NAT is configured correctly for your topology — the wrong public-IP/NAT setting is the most common cause of a phone that registers locally but not remotely, or vice versa. Also check Security/Firewall — a phone's IP could be legitimately banned by fail2ban if it retried a wrong password enough times; the ban is a real firewall rule, not just a flag, and needs to be cleared there.

Inbound calls to a number don't ring anywhere.

Confirm the DID actually has an inbound route pointed at it (Inbound routes, or Inventory → DIDs) and that the destination it points to is itself correctly configured (an extension that's registered, an IVR with a valid default option, etc.). Route Checker will show you the full resolved path for that number in one place. If routing looks correct but nothing rings, check Trunks — the call may never be reaching the box at all if the trunk itself shows unregistered.

Outbound calls fail immediately, or after a delay.

An immediate failure usually means no outbound route matched the dialed pattern, or the matching route's trunk is unregistered — check Outbound routes and Trunks. A failure after several seconds' delay, especially with a real carrier rejection code, usually means the call reached the carrier but was rejected (wrong outbound caller ID, a trunk authentication problem, or a carrier-side restriction) — Troubleshoot's SIP-trace tool will show you the carrier's actual rejection response.

Voicemail-to-email, report emails, or invite links aren't arriving.

These all go through your configured SMTP relay — check Settings → Email delivery (or, on multi-tenant, whether this tenant has its own configured or is meant to fall back to the box-wide default). A "Test connection" style check there will confirm whether the relay itself is reachable with the given credentials before assuming the problem is on the recipient's end.

A time condition is routing calls to the wrong branch, or a report shows the wrong time.

Business-hours evaluation and generated timestamps use your **configured company timezone** (Settings → Company), not the server's own operating-system clock — the two are often different (a cloud VM's OS clock is routinely UTC regardless of where the business actually is). If this is wrong, set the correct timezone under Settings → Company; there's no need to (and no supported way to) change the underlying server's own OS timezone to match — the box can genuinely host more than one company's own timezone at once on a multi-tenant install, so the fix always happens at the company-settings level, not the OS level.

The Dashboard's system-time clock (topbar) shows the wrong timezone.

It renders whatever is set under Settings → Company — the same value time conditions themselves use. If it looks wrong, the fix is the same as above: correct the configured company timezone, not the server clock.

A queue call isn't reaching every agent, or callback isn't working.

Confirm the agent is actually logged into the queue (agents can self-manage this from the Portal) and that their own extension is registered. Wallboard shows live queue state, including which agents are currently available. If callback specifically isn't firing, confirm it's enabled on that queue and that the scheduled-jobs timer that drives it (`run-queue-callbacks`) is running normally — `ringstack-cli status` is a quick way to check overall service health.

SMS isn't sending or receiving.

SMS rides your existing SIP trunk (RFC 3428 MESSAGE) — it only works if your trunk provider actually supports SMS-over-SIP delivery for that number. Not every carrier trunk does, and RingStack can't add support a carrier doesn't offer. Confirm with your carrier that SMS-over-SIP is enabled for the specific DID in question.

Multi-tenant: "This command needs --tenant" on a CLI command.

Most `ringstack-cli` commands operate on one tenant's own data on a multi-tenant install and need an explicit `--tenant <slug>` to say which one — run `ringstack-cli list-tenants` to see valid slugs. Commands that manage the shared control plane itself (tenants, global admins, licensing, HA, the box-wide email default) never take `--tenant` — see the CLI reference for which is which.

A license feature isn't unlocking after applying a license.

Run `ringstack-cli license-status` to see exactly what this box's currently-applied license carries (business name, feature list, expiry) as real JSON, not what you expect it to say. A license is locked to the specific machine it was issued for — check that "different machine" isn't the actual status before assuming the license itself is wrong.

Something changed and the system looks stale / out of sync.

`ringstack-cli reload` regenerates every config file from the current database state and reloads the affected Asterisk subsystems — the normal fix for "I know I saved this, but it doesn't seem to be taking effect." Settings → Dashboard also shows a live config-sync status pill if a recent automatic regeneration actually failed.

Something is badly broken and you want a known-good state fast.

`ringstack-cli restart-all` restarts every core service (Asterisk, the backend, nginx, and more) in one motion, waiting briefly for active calls to drain first unless `--force` is passed. It deliberately does not restart the database engine unless you explicitly pass `--include-db`, since that's a different, bigger kind of risk.

Getting more help

Stuck on something specific? The Troubleshoot assistant can trace a real call's dialplan, check registration state, read recent Asterisk errors, and pull a real SIP trace for you — turn it on under AI integration if it isn't already.

Found something that looks wrong, or have a suggestion? Leave it through the feedback bubble in the console — it's reviewed, not acted on automatically, so nothing changes without someone looking at it first.

Need someone else's hands on the box itself? **Support access** opens a real, time-limited shell session for a vendor to help directly, without leaving standing access behind afterward — see Email delivery & support access relays.