BrowserBox Customer Guide

DOSAYGO

Current as of BrowserBox v20.6.2
November 1, 2026

Status: Provisional customer reference. Supported commands, environment variables, login-link parameters, and operational patterns are subject to change in future BrowserBox releases.

Contents

1 Purpose **5**

2 The Native Freelang Architecture **5**

3 Installation and Customer Resources **6**
3.1 Official install entrypoints . . . 6
3.2 Windows . . . 6
3.3 Updates . . . 7
3.4 Suggested first run . . . 7
3.5 Prerequisites . . . 7
3.6 Network requirements . . . 8
3.7 Operating behind an HTTP forward proxy . . . 8
3.8 Audio prerequisites on headless hosts . . . 9
3.9 Selecting or pinning Chrome . . . 9
3.10 Full machine setup (`--full-install`) . . . 9
3.11 Support and public references . . . 10

4 Fast Start **10**
4.1 Standard local workflow . . . 10
4.2 Core commands . . . 11
4.3 Desktop app (`bbx gui`, beta) . . . 11
4.3.1 Session and Connection: start and share a browser . . . 12
4.3.2 Fleet: manage a Linux session pool . . . 12
4.3.3 Policy and All commands . . . 15
4.4 Automation and machine-readable output . . . 15

5 Customer Customization Surfaces **17**
5.1 Login-link URL parameters . . . 17
5.2 Homepage selection . . . 17
5.3 Declared startup tabs . . . 18
5.4 Cross-origin iframe compatibility (OOPIF) . . . 18

5.5 Printing . . . 18
5.6 Custom cursor . . . 19
5.7 UI theme override . . . 19
5.8 Blocked-page customization . . . 19
5.8.1 Changing only the support contact . . . 19
5.8.2 Replacing a page . . . 19
5.8.3 Requirements an installed page must meet . . . 20
5.8.4 Authoring notes . . . 20
5.9 Idle shutdown after disconnect . . . 21
5.10 Optional shutdown while clients remain connected . . . 21

6 Customer Policy Surface **22**
6.1 What policy controls . . . 22
6.2 Canonical policy fields . . . 22
6.3 Allowed values and defaults . . . 23
6.3.1 Binary feature controls . . . 23
6.3.2 Enumerated policy fields . . . 23
6.3.3 Identity and access configuration . . . 24
6.3.4 List-valued policy fields . . . 24
6.3.5 Where customers can verify allowed values . . . 24
6.4 Sensitive actions BrowserBox can allow or deny . . . 25
6.5 Navigation, browser-internal and private-network policy . . . 25
6.5.1 Navigation modes . . . 26
6.5.2 How allowlist entries match . . . 26
6.5.3 Private-network controls . . . 27
6.6 Certificate-error policy . . . 27
6.7 Browser chrome policy modes . . . 27
6.8 Locking kiosk mode . . . 28
6.9 Per-destination feature controls . . . 29
6.10 Server policy versus embedder policy . . . 29
6.11 Customer policy CLI . . . 29
6.11.1 Useful commands . . . 30
6.11.2 Recipe: global policy with a per-user tightening . . . 30
6.12 Operational visibility . . . 31

7 Embedding and Origin Allowlisting **31**
7.1 Server-side embedding allowlist . . . 31
7.2 Canonical embedder controls . . . 32
7.3 First-load tab cleansing . . . 32
7.4 Embedder media permissions . . . 33
7.5 Embedder session unload warning . . . 33
7.6 Remote page beforeunload behavior . . . 34
7.7 Modal lifecycle and dismissal API . . . 34

8 Setup and Backend Modes **36**
8.1 `bbx setup` . . . 36
8.2 Login-token lifecycle . . . 37
8.3 Connection types keep their own configuration . . . 37

8.4 Persistent configuration overrides (`user.env`) . . . 37
8.4.1 Recommended high-throughput profile . . . 38
8.4.2 Cross-origin iframe instrumentation . . . 39
8.5 Reverse proxy and HTTP backend . . . 39
8.6 Reverse proxy: preserving service origins (audio and DevTools) . . . 39
8.6.1 The DNS facade approach (recommended) . . . 39
8.6.2 Direct multi-port exposure (alternative) . . . 41
8.6.3 Audio connects but is silent . . . 41
8.6.4 Minimal mode (no audio or DevTools) . . . 42
8.7 External TLS hint . . . 42

9 Tunnel and Network-Oriented Commands 42
9.1 `bbx ng-config` . . . 42
9.2 `bbx ng-run` . . . 43
9.3 `bbx tor-run` . . . 43
9.4 `bbx zt-run` . . . 44
9.5 `bbx cf-run` . . . 44

10 Cross-User Execution (`--for`) 44
10.1 Requirements . . . 44
10.2 Supported commands . . . 44
10.3 What happens . . . 44
10.4 Example workflow . . . 45
10.5 Combining with certificate override . . . 45

11 Zeta Mode and `hosts.env` 45

12 Fleet: A Pool of Ephemeral Sessions (`bbx fleet`) 45
12.1 The mental model . . . 46
12.2 The three layers . . . 46
12.3 Connector and co-browsing semantics . . . 46
12.4 Requirements . . . 46
12.5 Privileged operator and licence . . . 47
12.6 Initialization . . . 47
12.7 DNS . . . 48
12.8 Certificates . . . 48
12.9 Forward-proxy defaults for Fleet seats . . . 49
12.10 Routing modes . . . 49
12.11 Acquire and release . . . 50
12.12 Keeping a small warm pool . . . 50
12.13 Calling Fleet from a web application . . . 51
12.14 Clean-slate security . . . 52
12.15 Operational commands . . . 52
12.16 Automatic dead-runtime recovery . . . 52
12.17 Security guidance . . . 53
12.18 Reference architecture . . . 54

13 Docker Deployments (`bbx docker`) 54
13.1 Quick start . . . 54
13.2 Docker engines and platforms . . . 55
13.3 Commands . . . 55
13.4 Setup options . . . 56
13.5 Ports and TLS modes . . . 56
13.6 Data, licence and updates . . . 57
13.7 Running the image on your own infrastructure . . . 57
13.8 Hyperscale in the desktop app (beta) . . . 58
13.9 Docker environment variables . . . 61

14 TLS and Certificates 61
14.1 BrowserBox-managed TLS . . . 61
14.2 Customer-managed certificates . . . 62

15 Customer-Relevant Environment Variables 63
15.1 Core customer-facing variables . . . 63
15.2 Examples . . . 68

16 Licensing and Occupancy Visibility 69
16.1 Available today . . . 69
16.2 When every seat is in use . . . 69
16.3 Ticket upgrade on first certification . . . 69
16.4 What `bbx vacancy` means . . . 69
16.5 What is not surfaced here . . . 70

17 Practical Recipes 70
17.1 Embedded BrowserBox without the standard UI . . . 70
17.2 BrowserBox behind a reverse proxy . . . 70
17.3 Check whether a seat is currently available . . . 70

18 Transport and Streaming 70
18.1 WebRTC (default) . . . 70
18.2 WebSocket fallback . . . 71
18.3 Capture profile and backpressure . . . 71
18.4 Streaming diagnostics . . . 71
18.5 Remote interaction . . . 72

19 Flipbook Recording 72
19.1 Quick start . . . 72
19.2 How it works . . . 72
19.3 Output structure . . . 73
19.4 Frame normalization . . . 73
19.5 Cloudflare Pages deployment . . . 73
19.6 Relevant environment variables . . . 73
19.7 Direct site generation . . . 74

1 Purpose

This document combines the customer-facing BrowserBox setup guide and the customer-relevant configuration reference into one printable manual. It is intended to answer the questions customers usually ask first:

- How do I install BrowserBox on macOS, Linux, or Windows?
- What is the native Freelang backend, and what does it guarantee?
- How do I set BrowserBox up and start it?
- Is there a desktop app, and how do I automate `bbx`?
- What can I customize?
- How do reverse proxy, HTTP mode, and certificates work?
- How do I embed BrowserBox safely in my own application?
- How do I operate Fleet and keep a small pool of ready sessions?
- What do the tunnel-oriented commands mean?
- What licensing visibility exists today?

2 The Native Freelang Architecture

BrowserBox v20 uses a native backend compiled from Freelang. Freelang is an ahead-of-time systems language designed to keep program behavior, authority, and resource ownership explicit and inspectable. It produces a standalone machine-code executable; the main BrowserBox request, control, session, and rendering server does not run inside Node.js.

The customer command layer remains `bbx` on macOS and Linux and PowerShell `bbx` on Windows. The commands, configuration directory, login links, policy files, and update workflow in this guide are unchanged by the backend implementation. BrowserBox selects the supported backend as part of the signed release; customers do not need a runtime backend switch.

Freelang gives the BrowserBox backend a smaller and more explicit execution surface:

- Arrays, byte buffers, strings, object fields, and runtime values carry checked bounds, sizes, shapes, or type tags. Freelang source has no general pointer arithmetic, raw pointer casts, or user-facing in-process foreign-function interface.
- Managed heap objects use a non-moving collector. References remain stable while unreachable storage is reclaimed and reused. Allocation failures and corrupt runtime invariants stop at explicit checks instead of continuing with undefined memory access.
- Expected operating-system and network failures travel through declared *chaos* outcomes. Broken program invariants use a separate fatal path, so ordinary connection failure is not confused with memory or state corruption.
- Complicated host capabilities are kept behind narrow process boundaries. Chrome remains a separate browser process; TLS, WebRTC, document, audio, and other platform facilities use supervised components with bounded protocols and explicit lifecycle ownership.
- Release artifacts are stripped and tied to signed manifests and exact build provenance. This reduces dependency and inspection surface and makes unexpected replacement detectable during install and update.

These properties reduce common memory-corruption, dependency, and lifecycle risks. They are engineering guarantees at specific boundaries, not a claim that BrowserBox, Chrome, an operating system, or a sidecar can contain no defect. Logic errors remain possible, and external components retain their own security models. BrowserBox therefore keeps policy enforcement, bounded input handling, process isolation, release verification, and regression gates in addition to the language-level checks.

The compiler used to build BrowserBox is implemented with a separate bootstrap toolchain, but that toolchain is not shipped as the BrowserBox server runtime. Full machine setup may still provision Node.js for remaining auxiliary install or service tasks. This does not put the main Freelang backend inside a Node.js process or restore the former server dependency tree.

3 Installation and Customer Resources

3.1 Official install entrypoints

BrowserBox installation guidance is published at `https://browserbox.io`. The current public install entrypoints are:

- macOS and Linux: `curl -fsSL https://browserbox.io/install.sh | bash`
- Windows: `irm https://browserbox.io/install.ps1 | iex`

Linux is supported on x86-64 and ARM64 (aarch64); the installer selects the matching native binary. The Unix installer requires `unzip` and installs it when it is missing. This guide documents the bash `bbx` command on macOS and Linux. Windows is covered in Section 3.2.

3.2 Windows

The Windows PowerShell `bbx` supports the everyday commands: `setup`, `start`, `stop`, `restart`, `status` (including `--json`), `logs`, `certify`, `activate`, `vacancy`, `update`, `use-chrome`, `uninstall`, `gui`, and the connection commands `cf-start`, `tor-start`, `zt-start`, `ng-start`, `ng-config`, and `win9x-start`. Both PowerShell-style parameters (for example `-Port 8888`) and the Unix-style `--port 8888` forms are accepted.

Windows runs one BrowserBox instance per machine. The following are Linux- or macOS-only and are refused on Windows with an `unknown or unsupported command` message before BrowserBox starts: `fleet`, `start-as`, `stop-user`, and `--for`. Remote audio is not available on Windows; `bbx status` reports audio as disabled.

Windows-specific behavior worth knowing:

- Cloudflare, Tor, and nginx tools are downloaded on first use into `%LOCALAPPDATA%\browserbox\tools`. Installing ZeroTier One requires an Administrator prompt.
- `bbx ng-start` serves a single origin on port 443; zeta mode is not used on Windows.
- `bbx stop` also stops the Cloudflare, Tor, and nginx processes that `bbx` started.
- Services started over SSH keep running after the SSH session disconnects.
- `bbx uninstall` removes BrowserBox but keeps your configuration and certificates.

3.3 Updates

`bbx update` and the background update check discover the latest public stable release at https://dl.getbrowserbox.com/latest.json, then download release files from that mirror. Both discovery and downloads fall back to GitHub if the mirror is unavailable. Every file is verified against the signed release manifest whichever source it came from; a file from the mirror that fails verification is fetched again from GitHub. The first-time installers download from GitHub.

The Unix wrapper records lookup source, cache age, HTTP status, curl exit code and whether a proxy was exported in `~/.config/dosaygo/bbpro/update-check.log`. Proxy URLs and tokens are not logged. Previous lookup caches are retained as diagnostic evidence and are not treated as fresh release selections. Automatic updates may move to an older version when the live release selection deliberately rolls back a faulty release.

Updates are applied in two phases. A new release is downloaded and verified in the background, then installed at the start of your next `bbx` command, which re-runs on the new version. Installed executables are replaced atomically, so a command that is already running never sees a partially written file.

Variable	Effect
`BBX_NO_CDN=1`	Use GitHub for discovery and downloads. Use when outbound policy permits GitHub but not the mirror.
`BBX_ASSET_BASE`	Base URL of your own release mirror, with `latest.json` and `<tag>/<file>`. Files are still verified against the signed manifest.
`BBX_NO_UPDATE=true`	Disable updates, including an explicit `bbx update`. Unset it in the shell you upgrade from when you are ready to move to a new release.

Permit outbound HTTPS through your proxy to `dl.getbrowserbox.com` on TCP 443 for the normal public stable update path. For GitHub fallback, also permit `api.github.com`, `github.com` and release download redirects to `release-assets.githubusercontent.com`. Alternatively, set `BBX_NO_UPDATE=true` and upgrade through your own change process. Export proxy assignments in `user.env`; the Unix wrapper also exports its configured proxy variables to child processes.

3.4 Suggested first run

After installation, the normal customer workflow is:

```
bbx certify YOUR_KEY
bbx setup -p 8888
bbx start
```

If you are evaluating BrowserBox, obtain your license through https://browserbox.io. Commercial licensing and evaluation paths are surfaced there, and the site links through to the current purchase flow.

3.5 Prerequisites

BrowserBox requires a Chrome-family browser (Google Chrome or Chromium) installed on the host machine. If Chrome is not found, runtime commands (`start`, `run`, `ng-run`, etc.) will refuse to launch and print:

```
Chrome/Chromium is not installed.
BrowserBox requires a Chrome-family browser to run.
```

```
Install it with:  browserbox --full-install <hostname> <email>
```

3.6 Network requirements

BrowserBox validates its licence continuously while running (every 331 seconds), not only at activation time. Each licensing service has an independent fallback, so the host must be able to reach all four of the following endpoints over TCP 443:

- `https://license.dosaygo.com` and its fallback `https://license2.dosaygo.com`
- `https://master.dosaygo.com` and its fallback `https://master2.dosaygo.com`

Every request goes to the primary first. BrowserBox uses the fallback only when the primary cannot be reached (DNS, connection, TLS, or timeout failures, or a gateway error), so a licence remains usable during a primary outage. Validation is identical on either endpoint.

These endpoints may be reached directly or through a configured HTTP forward proxy. BrowserBox still performs the complete signed ticket, seat, and challenge-response validation when a proxy is used; proxy routing does not disable or weaken licence enforcement. If neither the primary nor the fallback endpoints can be reached by either route, the main service stops and restarts repeatedly, and `bbx logs` records errors such as:

```
[PKI] Error fetching public key: Fetch to https://license.dosaygo.com/keys/root timed out
Error: Local certificate validation failed.
```

On networks where egress is restricted, permit all four destinations before deploying. Routine licensing detail is not logged by default; set `BBX_DEBUG_LICENSE=true` and restart when support asks for licensing diagnostics. If direct egress is prohibited, configure the forward proxy as described below.

3.7 Operating behind an HTTP forward proxy

BrowserBox honours the standard upper- and lower-case proxy variables: `HTTP_PROXY`, `HTTPS_PROXY`, `ALL_PROXY`, and their lower-case forms. `NO_PROXY` and `no_proxy` exclude destinations from proxy routing. These are operating-system conventions, not BrowserBox-specific variables.

Set the proxy variables in the environment that starts BrowserBox, or persist them in `user.env` (Section 8.4). Also set both forms of `NO_PROXY` so loopback and internal traffic from BrowserBox and its dependencies stays local:

```
export http_proxy="http://proxy.example.com:3128"
export https_proxy="$http_proxy"
export no_proxy="localhost,127.0.0.1,::1,10.0.0.0/8,.internal.example.com"
export NO_PROXY="$no_proxy"
```

BrowserBox's own loopback health checks explicitly bypass forward proxies. A complete `NO_PROXY` value remains recommended because package managers, Chrome, and other host tools may independently honour the same proxy environment.

The service orchestrator propagates the active `BBX_*`, proxy, certificate-authority, and sound-session variables to every BrowserBox service and its restart guardian. Proxy credentials do not need a second BrowserBox-specific configuration surface.

If the corporate proxy performs TLS inspection, add its trusted CA before starting BrowserBox rather than disabling certificate validation:

```
export FLXTLS_CA_BUNDLE="/etc/pki/ca-trust/source/anchors/corporate-proxy.pem"
bbx start
```

`FLXTLS_CA_BUNDLE` is read when the native BrowserBox runtime starts, so restart BrowserBox after changing it. Existing deployments may retain `NODE_EXTRA_CA_CERTS` as a migration alias, but new native configuration should use `FLXTLS_CA_BUNDLE`.

3.8 Audio prerequisites on headless hosts

The audio service captures sound from a PulseAudio or PipeWire server running in the *service user's* session. On a headless server that session does not exist by default. BrowserBox checks this dependency during startup: the main service continues running, while `bbx status` reports audio as `Ready`, `Degraded`, or `Disabled` with an actionable detail.

On systems using PipeWire's PulseAudio compatibility service, BrowserBox installs a native `pipewire-pulse` drop-in that creates the `channel` playback sink, the `rtp` capture sink, and the loopback between them. `bbx setup` selects `channel` for Chrome, while the audio service captures `rtp.monitor`. This configuration is capability-detected: systems using native PulseAudio keep the established `default.pa` path unchanged. The drop-in is installed by both a full installation and the command-install step run automatically by `bbx update`.

Ensure the service user has a durable session:

```
sudo loginctl enable-linger bbuser
```

Start BrowserBox as that user, from that user's own login session. Then verify:

```
pactl info                      # should print a Server String
pactl list short sinks          # at least one sink present
pactl list short sources        # should include rtp.monitor
```

While a page is playing sound, `pactl list short sink-inputs` should show an active stream. See Section 8.6.3 if it does not.

3.9 Selecting or pinning Chrome

BrowserBox normally discovers a compatible Chrome-family browser automatically. When support recommends a specific Chrome major for compatibility testing, use:

```
bbx use-chrome 149
```

The command resolves and installs the requested Chrome-for-Testing major, then persists the resulting `CHROME_PATH` in `~/.config/dosaygo/bbpro/user.env`. The selection therefore survives `bbx setup` and service restarts. You may also supply an executable path directly, for example `bbx use-chrome /opt/google/chrome/chrome`.

Pinning is a compatibility or rollback measure, not the first response to every startup failure. Before changing Chrome, run `bbx logs` and check for integrity, configuration, license, port, or certificate errors. To return to automatic browser discovery, remove the `CHROME_PATH` line from `user.env` and run `bbx restart`.

3.10 Full machine setup (`--full-install`)

On a fresh server without Chrome or other dependencies:

```
browserbox --full-install <hostname> <email>
```

- `<hostname>` — the domain or IP address (e.g. `localhost`, `bbx.example.com`, `198.51.100.42`).
- `<email>` — email used for BrowserBox terms acceptance and, when applicable, public-certificate issuance.

This installs Chrome, TLS tooling, the auxiliary dependencies described in Section 2, and the other OS-level packages BrowserBox needs. For non-interactive installs, the preferred environment variables are `BBX_INSTALL_HOSTNAME` and `BBX_INSTALL_EMAIL`. Legacy aliases `BBX_HOSTNAME`, `BBX_EMAIL`, and `EMAIL` are still accepted. If you run the standalone installer as `root` in a non-interactive environment, also set `BBX_INSTALL_USER` to the non-root account that should own the installation.

For non-interactive (scripted) installs, set `BBX_TEST_AGREEMENT=true` to auto-agree to the license agreement:

```
export BBX_TEST_AGREEMENT=true
export BBX_INSTALL_HOSTNAME="bbx.example.com"
export BBX_INSTALL_EMAIL="you@example.com"
browserbox --full-install "$BBX_INSTALL_HOSTNAME" "$BBX_INSTALL_EMAIL"
```

3.11 Support and public references

- Product site and license purchasing: https://browserbox.io
- Public GitHub repository: https://github.com/BrowserBox/BrowserBox
- Support email: support@dosaygo.com

4 Fast Start

4.1 Standard local workflow

```
bbx stop
bbx setup -p 8888
bbx start
```

When you run `bbx setup -p 8888`, port `8888` becomes the main BrowserBox service port. Related service ports are derived from the main port:

- audio: main port - 2
- docs: main port - 1
- devtools: main port + 1

BrowserBox writes the current login link to:

```
~/.config/dosaygo/bbpro/login.link
```

4.2 Core commands

Command	Purpose
`bbx setup`	Configure port, hostname, token, zeta mode, and backend mode.
`bbx start`	Start BrowserBox for the current user. `--port` and `--hostname` change and save the address (Section 8.3).
`bbx stop`	Stop BrowserBox for the current user.
`bbx restart`	Stop and relaunch BrowserBox using the connection type it was last started with.
`bbx status`	Check whether BrowserBox is reachable and running. `--json` prints a machine-readable result.
`bbx logs`	View BrowserBox logs.
`bbx certify`	Validate the current license and reserve a seat when applicable.
`bbx vacancy`	Show total, occupied, vacant, leased, and reserved seat counts, plus any local reservation metadata on the machine.
`bbx update`	Update BrowserBox to the latest or a specific release (Section 3.3).
`bbx use-chrome`	Install and select a specific Chrome version.
`bbx gui`	Install and open the BrowserBox desktop app (Section 4.3).
`bbx policy`	Inspect, validate, and set browsing policy (Section 6.11).
`bbx fleet`	Allocate and release clean-slate sessions from a Linux user pool (Section 12).
`bbx ng-config`	Print, validate, or atomically apply the four-service nginx facade configuration.
`bbx ng-run`	Run the nginx-oriented workflow.
`bbx tor-run`	Run BrowserBox with Tor integration.
`bbx zt-run`	Run BrowserBox on a ZeroTier network.
`bbx cf-run`	Run BrowserBox through a Cloudflare tunnel.
`bbx uninstall`	Remove BrowserBox, including the desktop app.

Each `*-run` command also has a `*-start` spelling (for example `bbx cf-start`); the two are equivalent. `bbx --help` lists the commands, and `bbx --help-json` prints the complete command catalogue, including every flag, as JSON.

`bbx logs` displays the supervised BrowserBox service logs. Process custody and restart behavior are internal runtime responsibilities; normal operation does not require a separate process-manager command.

4.3 Desktop app (`bbx gui`, beta)

BrowserBox includes a desktop app for people who prefer buttons to the command line. It is carried inside the BrowserBox binary, so there is nothing separate to download:

```
bbx gui              # install if needed, then open
bbx gui --reinstall  # reinstall the app from the current binary
```

The first run installs the app for the current user and prints where it lives, so it can be pinned: `~/Applications/BrowserBox.app` on macOS, a `browserbox.desktop` launcher on Linux, and a **BrowserBox** Start-menu shortcut on Windows (`bbx gui -Reinstall` in PowerShell). On Linux, run it from a desktop session; it needs a display.

The app is a front end to the same `bbx` commands described in this guide. Every action shows the exact command it will run. It has six tabs:

Tab	What it does
Session	Shows whether BrowserBox is running and how it is reachable. Start, open, restart, or stop it, and copy the login link. The token itself is never displayed.
Connection	Choose how BrowserBox is reached: Direct, Cloudflare, ZeroTier, Tor, Nginx, or Legacy (Windows 9x mode), then launch it.
Fleet	A dashboard for a Fleet seat pool: capacity, public-route health, per-seat status, and pool actions such as new session, doctor, reconcile, and apply routes. Linux only; disabled on macOS and Windows.
Policy	View the effective browsing policy, validate it, check an action, or install or reset a policy file.
Hyperscale	Manage named local Docker deployments, or read assisted Google Cloud Run setup guidance.
All commands	A searchable catalogue of every `bbx` command with a form for its options.

The **Hyperscale** tab is available directly; it does not require an experiment switch. See Section 13 for deployment setup and lifecycle commands.

Software Update... in the File menu runs `bbx update`. The desktop app is in beta; the `bbx` command line remains the complete, supported interface.

4.3.1 Session and Connection: start and share a browser

Open **Connection**, choose the connection type, and fill in its address and options. Labels distinguish saved values, unsaved edits, CLI defaults, and required fields. **Reload saved** discards edits; **Launch** applies the form. The Session tab's Start button uses that same selected connection. Changing a form alone does not launch a browser.

In **Session**, follow Activity while startup completes, then choose **Open BrowserBox** or **Copy login link**. A saved link and a reachable service are separate observations: a link can appear before startup verification finishes. Share the full login link only with intended users; it grants access to the session.

The screenshots below use the actual desktop renderer with example data. Hostnames, deployments and seat states are illustrative; available actions depend on the host platform and permissions.

Open a recent task or use **View** → **Output window** to inspect command output. Closing the app offers **Close GUI only**, **Stop and close**, and **Keep open**. Close GUI only leaves independent BrowserBox services running and is unavailable while the window owns a non-status command. Use Stop and close to end an owned foreground tunnel through the normal stop flow.

4.3.2 Fleet: manage a Linux session pool

On a Linux Fleet host, open **Fleet** and use **Refresh** to read pool capacity, routing health and seat state. Select a seat to inspect its allocation. An active seat offers **Open**, **Copy link**, **Check**, and **Release...** when available; a free seat can start a session. **New session** allocates from the pool. Release ends that allocation and returns its seat to the pool.

Initialize... configures a pool; **Doctor** diagnoses it. Reconcile, Repair, Reap expired and Apply routes are administrative operations: read the proposed command and confirmation before

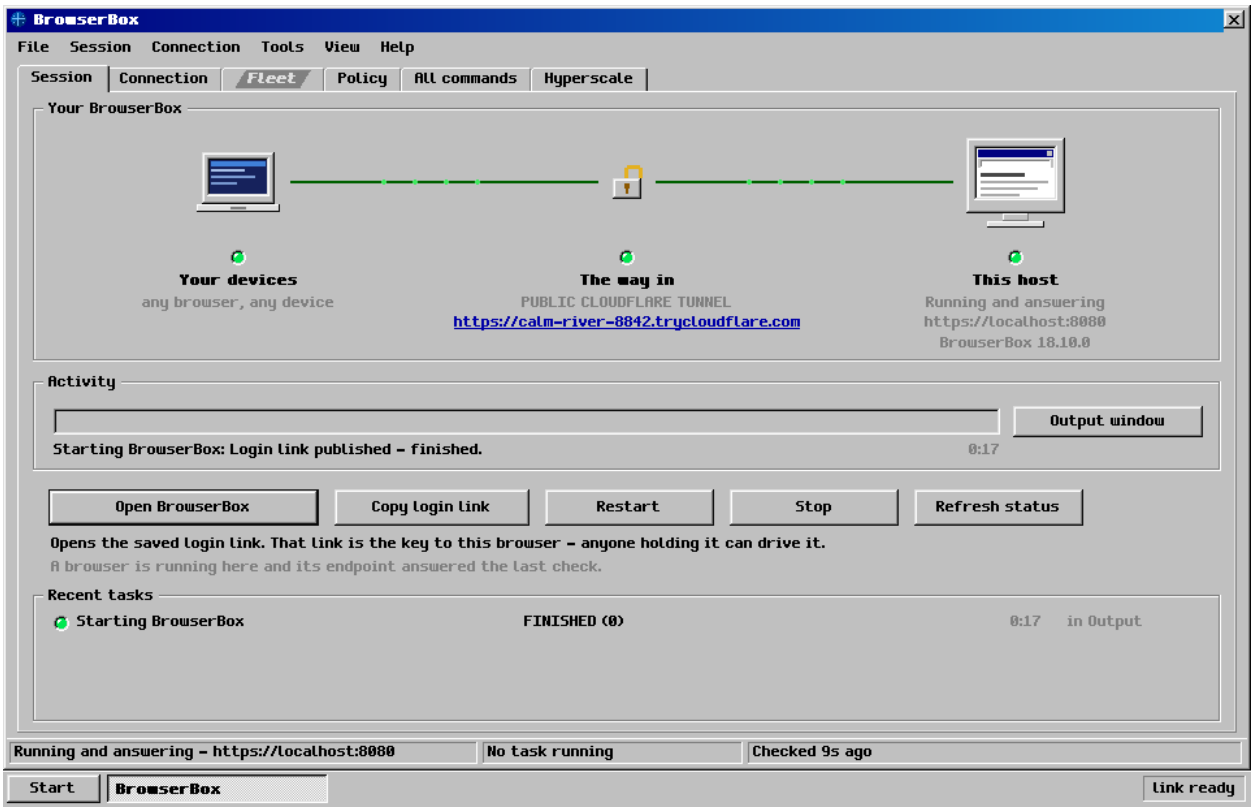


Figure 1: Session: connection route, service state, Activity and recent tasks.

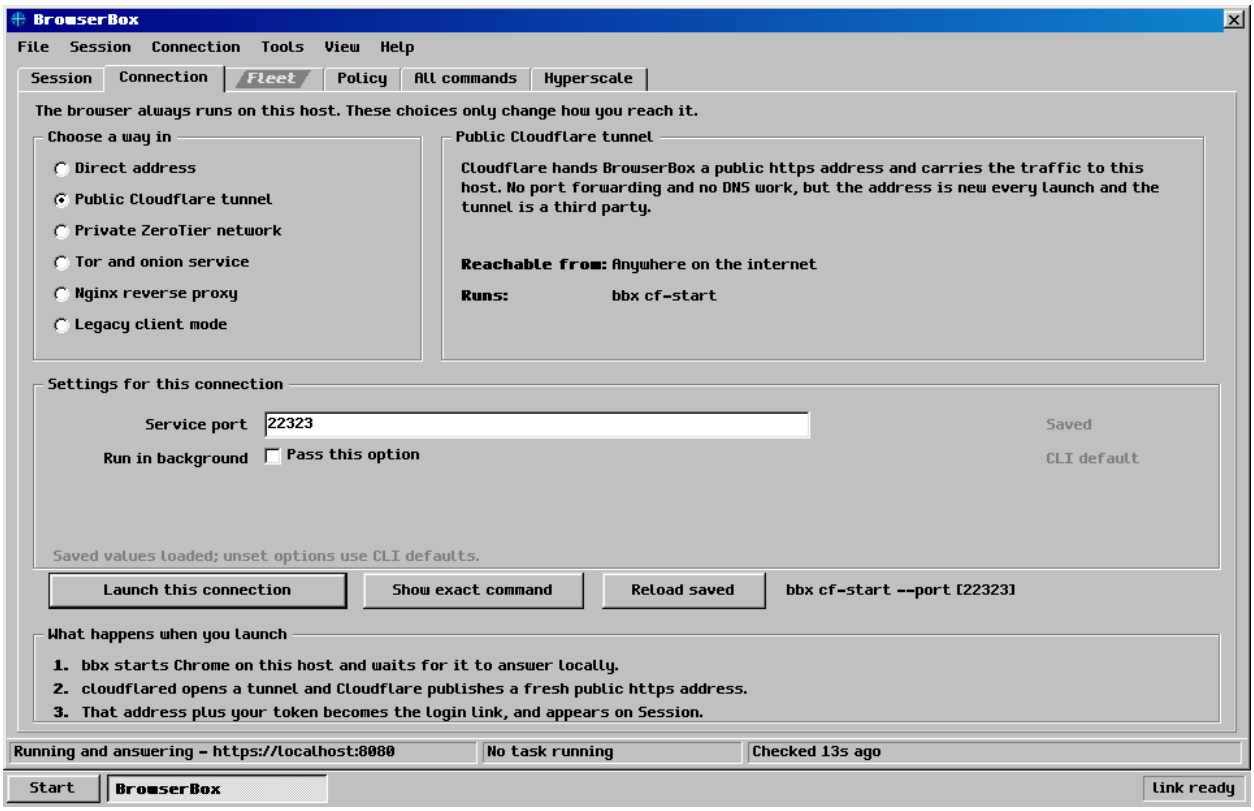


Figure 2: Connection: choose a route, review saved settings, then launch.

proceeding. Privileged actions require the corresponding host permissions. Fleet is disabled on macOS and Windows.

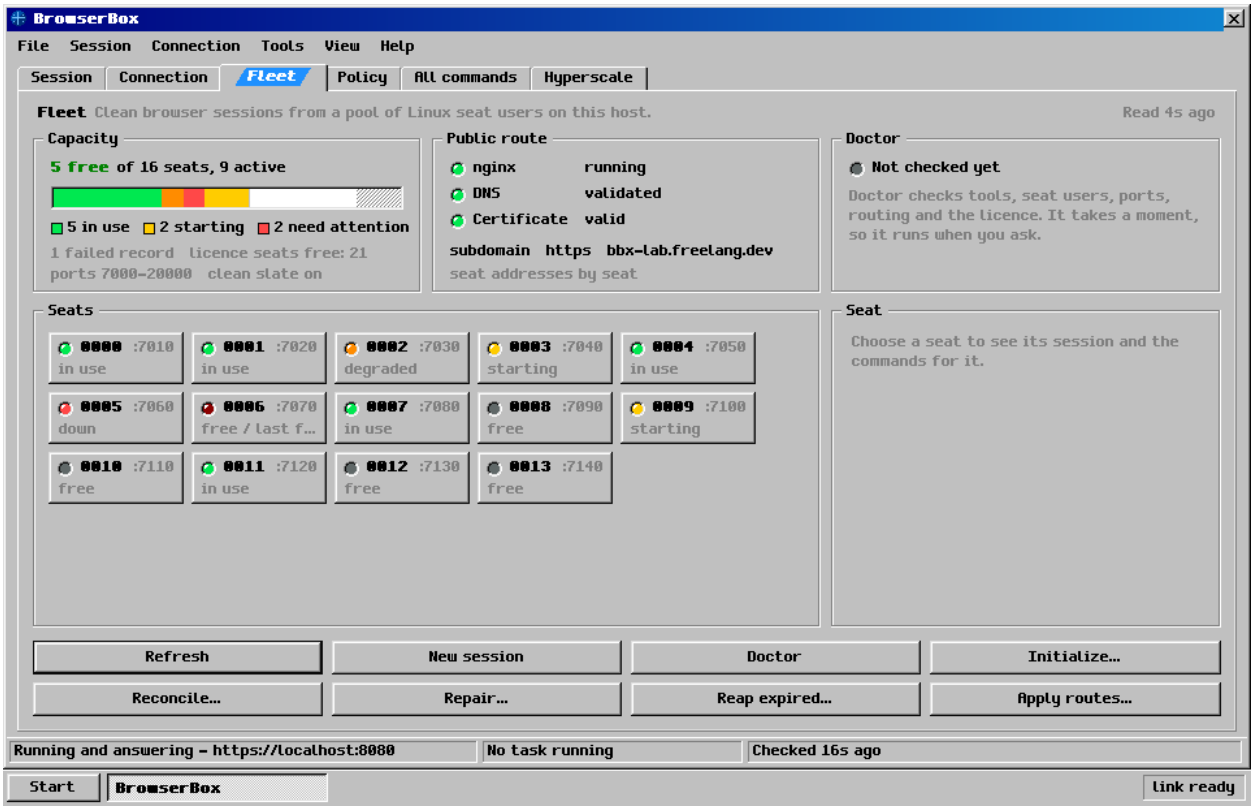


Figure 3: Fleet: Linux pool capacity, public routing health and selectable seats.

4.3.3 Policy and All commands

Open **Policy** to read the effective rules and their source locations, validate the policy, or check an action before applying a change. Installing or resetting a policy changes the machine's configuration; review the command and selected file first.

Use **All commands** to search the complete command catalogue and fill in options that do not have a dedicated panel. Review the command preview before running it. Commands requiring interactive terminal input still need a terminal.

4.4 Automation and machine-readable output

`bbx` keeps actionable results on standard output and informational messages (progress, update checks, policy footers) on standard error, so scripts can capture results cleanly.

```
bbx status --json                    # one JSON document on stdout
bbx status --out /path/status.json   # same, written atomically to a file
bbx --help-json                      # full command catalogue (schema bbx.help/1)
bbx --output-log /path/new.log -- start   # send a command's output to a new file
```

`bbx status --json` reports `running`, `hostname`, `scheme`, `main_port`, `version`, the audio state, and the active connection type. It never includes the login token. `--output-log` takes an absolute

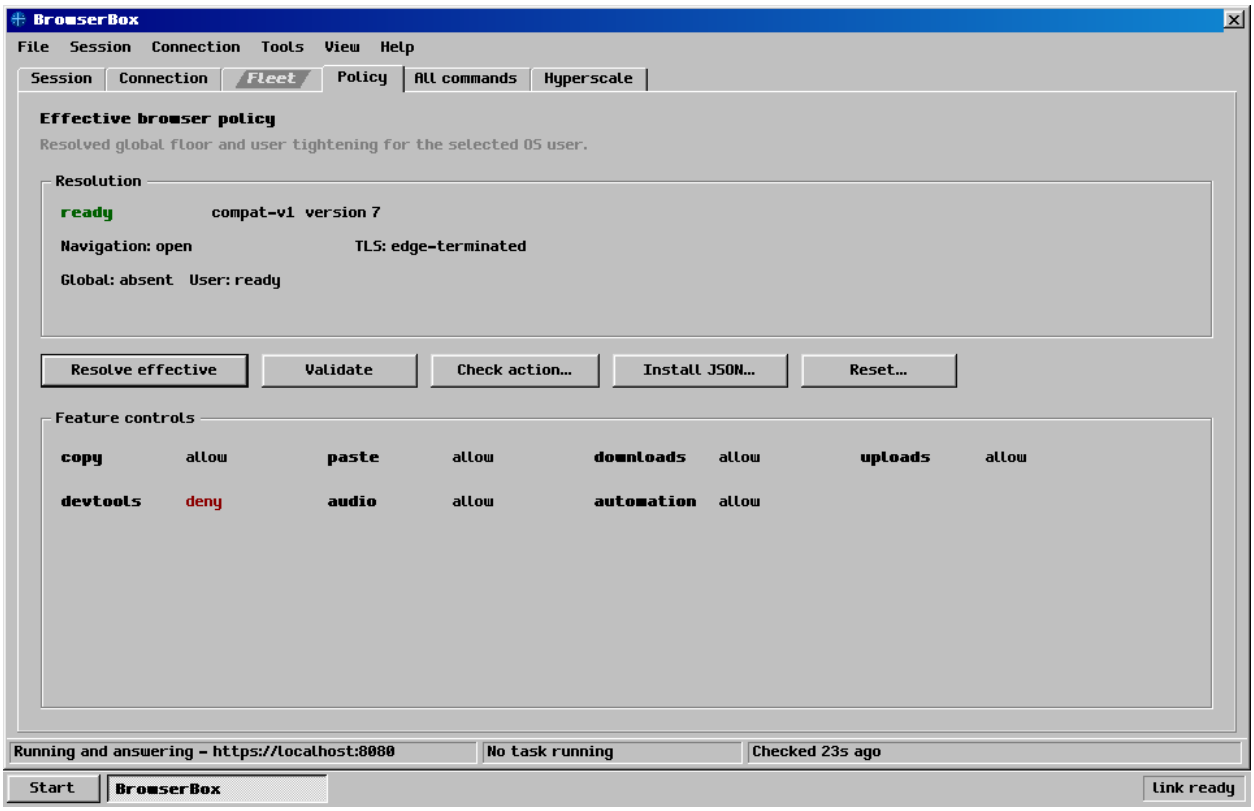


Figure 4: Policy: inspect effective browser controls and run policy operations.

path, writes both output streams to that new owner-only file, and refuses to overwrite an existing file.

Set `BBX_PROGRESS_EVENTS=1` to have long-running commands print one-line progress markers such as `@bbx-progress ready` and failure markers such as `@bbx-failure license-refused`. These are intended for wrappers and orchestration tools that need to show progress without parsing human-readable output.

5 Customer Customization Surfaces

5.1 Login-link URL parameters

BrowserBox generates login links with query parameters that affect customer-visible behavior.

Parameter	Meaning
`ui=false`	Hides the standard BrowserBox chrome for embedding or kiosk-style use.
`url=`*`URL or JSON payload`*	Opens target pages on first load. Use a bare absolute URL or a JSON string for one starting URL, or a JSON array of strings for multiple starting URLs. URL-encode the complete value when placing it on the login link.
`mid=`*`machine-id`*	Infrastructure routing affinity in multi-machine deployments. This is not a user identity or a seat identifier.
`uiTheme=`*`theme`*	Requests a UI theme when the server accepts that theme value.

For one starting page, the simplest value is a bare URL such as https://example.com. A JSON string such as "https://example.com" is also accepted. To open multiple starting tabs, use a JSON array such as ["https://example.com","https://docs.example.com"]. URL-encode the complete value before appending it as `url=...`; do not encode or modify the login token itself.

When reopening a login link for a running session, the `url=` parameter will intelligently reuse and navigate an existing suitable tab rather than creating a duplicate tab. This makes it fully compatible with single-tab topologies such as `BBX_MAX_TABS=1`.

The `ui=` parameter sets the starting visibility only. It is not a lock: a user can change it, for example by editing `ui=` in the address bar and reloading. To keep the chrome hidden whatever the link or the user's browser says, use the server policy in §6.8.

5.2 Homepage selection

Use either of these variables:

```
export BBX_HOME_PAGE="https://intranet.example.com"
# or
export BBX_DEFAULT_HOME_PAGE="https://intranet.example.com"
```

By default, BrowserBox asks Chrome to restore tabs from the previous browser session. If no usable page tab is restored, BrowserBox opens the configured home page. To preserve the browser profile while always beginning with the home page instead of restored tabs, set:

```
export BBX_RESTORE_LAST_SESSION=false
export BBX_HOME_PAGE="https://intranet.example.com"
```

5.3 Declared startup tabs

Use `BBX_STARTUP_TABS` to declare the exact tab topology for the initial BrowserBox start. The value is a non-empty JSON array of absolute URLs:

```
export BBX_STARTUP_TABS='["https://intranet.example.com"]'
```

This starts BrowserBox with exactly one page tab without restoring tabs from the previous Chrome session. It preserves cookies, local storage, preferences, and other browser-profile data. It does not restrict later navigation or prevent the user from opening more tabs.

Multiple initial tabs are also supported:

```
export BBX_STARTUP_TABS='["https://intranet.example.com", "https://docs.example.com"]'
```

The number of declared startup tabs must not exceed `BBX_MAX_TABS`. Navigation policy still applies to every declared URL. When set, `BBX_STARTUP_TABS` takes precedence over `BBX_HOME_PAGE` and `BBX_RESTORE_LAST_SESSION` for initial startup.

The value must be a valid `http:` or `https:` URL. Invalid values fall back to https://duckduckgo.com.

5.4 Cross-origin iframe compatibility (OOPIF)

BrowserBox enables out-of-process iframe (OOPIF) support by default. Modern Chrome may place a cross-site iframe in a separate renderer process; BrowserBox attaches to that target so cursor support, page integrations, and supported injected capabilities continue to work inside the frame. This also applies recursively to nested cross-site frames.

The same attachment policy safely releases dedicated and shared Web Workers without treating them as tabs. Worker-dependent applications, including PDF.js viewers, therefore continue to run while OOPIF support remains enabled. If a worker-dependent page stalls on an older BrowserBox release, upgrade before disabling OOPIF support permanently.

If a site exhibits a compatibility or stability problem after an upgrade, an administrator can temporarily return to the previous behavior:

```
export BBX_OOPIF=false
bbx restart
```

For a persistent setting, add `BBX_OOPIF=false` to `~/.config/dosaygo/bbpro/user.env`, then restart BrowserBox. Disabling OOPIF support means BrowserBox will no longer attach to separately rendered cross-site iframe targets, so BrowserBox-provided cursor and page integrations may not be available inside those frames. Include `bbx logs` output when reporting an OOPIF compatibility issue.

5.5 Printing

BrowserBox handles printing as an explicit download intent rather than opening a printer attached to the BrowserBox host. When a page calls `window.print()`, the built-in Chrome PDF viewer requests printing, or the user selects **Print** from the BrowserBox context menu, BrowserBox asks for confirmation. After confirmation, BrowserBox creates or retrieves a PDF and presents the existing download-ready dialog. The user downloads that PDF and prints it with a local printer.

Printing requires downloads to be allowed by the active BrowserBox policy. A print request from a separately rendered OOPIF is captured in that target; a request from a same-process iframe produces the printable document for its owning page target.

5.6 Custom cursor

```
export BBX_CUSTOM_CURSOR="/absolute/path/to/cursor.png"
bbx start
```

BrowserBox reads the local file at startup and uses it as the remote cursor image.

5.7 UI theme override

```
export BBX_UI_THEME_OVERRIDE=dark
bbx start
```

This sets the default BrowserBox UI theme for the login flow. A login-link `uiTheme=` parameter can also request a theme when accepted by the server.

5.8 Blocked-page customization

When BrowserBox refuses a navigation, it serves a page in place of the site. By default that page carries BrowserBox branding and directs the user to DOSAYGO support. Deployments that enforce their own allowlists will usually want the user directed to their own service desk instead.

5.8.1 Changing only the support contact

If the built-in wording and layout are acceptable and only the contact details are wrong, set one variable:

```
export BBX_SUPPORT_CONTACT="servicedesk@example.com"
bbx restart
```

An email address is rendered as a `mailto:` link. A value beginning `http://`, `https://`, `mailto:`, or `tel:` is used as the link target as given, so a ticketing form may be used instead:

```
export BBX_SUPPORT_CONTACT="https://servicedesk.example.com/new-request"
```

This variable reaches the two built-in pages marked in the table in §5.8.2, and nothing else. In particular it has no effect on a page you install, because an installed page is served exactly as written. The two mechanisms are alternatives rather than layers: either BrowserBox composes the page and reads this variable, or it serves your file and does not. A page you install should carry your contact details in its own markup.

5.8.2 Replacing a page

To replace a page entirely, install an HTML file named after the page id into a status-pages directory. Global pages are read from `/etc/browserbox/status-pages/`, which must be root-owned and world-readable; per-user pages are read from `~/.config/dosaygo/bbpro/status-pages/`. A global page takes precedence, so a page installed by an administrator cannot be replaced by the account the browser session runs as. A directory named by `BBX_STATUS_PAGES_DIR`, when set, is searched before both and is subject to the same ownership checks as a per-user directory.

File	Shown when	Built-in shows contact
blocked.html	A navigation is blocked by ad blocking or by a forbidden URL scheme.	Yes
blocked-policy.html	A navigation is blocked by policy, including `navigation.allowlist` and private-network rules (see §6.5).	Yes

The final column indicates whether the built-in version of that page displays the support contact, and therefore whether `BBX_SUPPORT_CONTACT` changes it.

Deployments enforcing an allowlist through BrowserBox policy should customize `blocked-policy.html`. That is the page a user reaches when they attempt to browse outside the allowlist.

`blocked-policy.html` falls back to `blocked.html`. Installing `blocked.html` alone therefore changes both; install `blocked-policy.html` as well only when policy denials should read differently from ad blocking.

```
sudo install -d -o root -g root -m 0755 /etc/browserbox/status-pages
sudo install -o root -g root -m 0644 blocked-policy.html \
  /etc/browserbox/status-pages/blocked-policy.html
```

Changes take effect within a few seconds, on the next blocked navigation. No restart is required.

Login and authentication screens, and the pages shown while the remote browser is starting or after it has failed, are part of BrowserBox itself and are not customer-overridable.

5.8.3 Requirements an installed page must meet

An installed page is used only if all of the following hold:

- it is a regular file, not a symbolic link
- it is not group-writable or world-writable
- it is owned by `root` for a global page, or by the account running BrowserBox for a per-user page
- the directory containing it meets the same ownership and permission requirements
- it is at most 256 KB

The most common installation error is copying the file as an ordinary user rather than as `root`. In `/etc/browserbox/status-pages/`, a file owned by any account other than `root` is ignored.

A file that fails any requirement is ignored, the built-in page is served in its place, and BrowserBox logs one line naming the file and the reason. A malformed or rejected page never changes whether a request is blocked, only what the user is told about it.

5.8.4 Authoring notes

The file is served exactly as written. BrowserBox applies no template syntax and performs no substitution of any kind — `BBX_SUPPORT_CONTACT` included — so a static page cannot name the address that was refused. The built-in policy page shows the address and the policy reason code; an installed page replaces that with whatever it contains.

Blocked pages are served inside the remote browser, at the origin of the address that was refused, and reach the user as streamed canvas output. The user's own browser does not parse the file. Three consequences follow:

- The file executes in the remote browser at that origin, and should be treated as trusted content. This is the reason for the ownership requirements above.
- `mailto:` and `tel:` links do reach the user. BrowserBox detects the external-protocol navigation, asks the user to confirm the request, and on approval opens the address on their own machine so their local mail client handles it. Write contact details as visible text as well, so they can be read and copied if the user declines the prompt.
- External stylesheets, fonts, and images are fetched by the remote browser and are subject to the same allowlist that produced the block. Inline CSS and embed images as `data:` URIs.

5.9 Idle shutdown after disconnect

```
export BBX_SHUTDOWN_IDLE_MS=7200000
bbx start
```

This controls how long BrowserBox waits after all clients disconnect before shutting down.

If you run with `BBX_MINIMAL_MODE="true"` and do not set `BBX_SHUTDOWN_IDLE_MS`, BrowserBox uses `BBX_MINIMAL_MODE_IDLE_MS` as the fallback idle shutdown duration.

5.10 Optional shutdown while clients remain connected

Native BrowserBox can also reclaim a connected but inactive runtime. This policy is disabled by default. The following example closes the runtime after 30 minutes without accepted activity and warns the user one minute beforehand:

```
export BBX_CONNECTED_IDLE_MS=1800000
export BBX_CONNECTED_IDLE_WARNING_MS=60000
export BBX_CONNECTED_IDLE_WARNING_MESSAGE="Your session will close in one minute unless
    activity resumes."
bbx start
```

Accepted pointer, keyboard, touch, navigation, tab, and modal commands renew the session only after the backend admits them. An advancing video in the active remote tab also renews it through Chromium's presented-video-frame callback. Ordinary frame capture, CSS animation, a blinking cursor, paused or stalled video, and muted or silent audio do not count as activity. During audio-only playback, the client sends a bounded, authenticated heartbeat only while it measures non-silent samples. Activity from any connected co-browser renews the shared session.

`BBX_CONNECTED_IDLE_WARNING_MS` defaults to half the configured inactivity period, capped at 60 seconds. Set it to zero to disable the warning. Dismissing the warning counts as activity and renews the deadline. The optional message is limited to 4096 UTF-8 bytes.

Expiry follows BrowserBox's normal graceful shutdown path. In Fleet, the existing monitor then confirms that the runtime is fully down and returns the allocation through the normal cleanup path. This is an inactivity policy, not an absolute session lifetime; a customer application must still enforce any maximum total session duration.

6 Customer Policy Surface

6.1 What policy controls

BrowserBox policy is the customer-facing runtime control surface that answers two distinct questions:

- Which sensitive actions are allowed or denied?
- Which browsing targets may the remote browser reach?

The canonical persisted policy file lives at:

```
~/.config/dosaygo/bbpro/policy/policy.json
```

BrowserBox normalizes and validates that file before applying it. If a field is omitted, BrowserBox fills in safe defaults for the current schema.

The allowed values listed in this section are based on BrowserBox's current runtime validation and normalization surface, not on a separate aspirational specification. In practice, customers should be able to trust this section because these values are the same ones BrowserBox validates at load time.

6.2 Canonical policy fields

The customer-relevant policy bundle currently includes these high-level surfaces:

Field	Meaning
`transport.certificateErrors`	Whether BrowserBox enforces certificate validation or explicitly allows self-signed / invalid certificates.
`navigation.mode`	Top-level navigation mode: `open`, `allowlist`, or `blocklist`.
`navigation.allowlist`	Allowed navigation targets when `navigation.mode=allowlist`. Supports exact host, exact origin, exact URL, and wildcard host patterns (§6.5.2).
`navigation.blocklist`	Denied navigation targets when `navigation.mode=blocklist`.
`navigation.internalAllowlist`	Exact browser-internal origins that may open. It defaults to empty. `chrome://inspect` and the other permanently reserved targets cannot be enabled.
`navigation.privateNetwork.mode`	Whether private/local targets are denied by default or controlled through an allowlist.
`navigation.privateNetwork.allowlist`	Explicit private/local destinations that may be reached, including exact hosts, IPs, and IPv4 CIDRs.
`featureControls.copy`	Controls copy actions, including modifier shortcuts and context-menu copy paths.
`featureControls.paste`	Controls paste actions, including text insertion into the remote browser.
`featureControls.downloads`	Controls download routes and archive-serving surfaces. When set to `deny`, the download is cancelled at the browser level before any data is written to disk, and the connected client receives a policy-denied notification.
`featureControls.uploads`	Controls upload routes and file input surfaces.

`featureControls.devtools`	Controls DevTools access.
`featureControls.audio`	Controls the audio sidecar service.
`featureControls.automation`	Controls the BrowserBox automation surface used by the webview API.
`featureControls.contextMenu`	Controls whether the remote-page context menu may open.
`featureControls.browserChrome`	Whether BrowserBox's tab bar and address bar are shown, hidden or user-toggleable. `always-hide` keeps them hidden whatever the login link, the address bar or an embedder asks for (§6.7).

6.3 Allowed values and defaults

The most important values customers need to know are listed here directly.

6.3.1 Binary feature controls

Unless stated otherwise, feature controls are binary decisions:

Field	Allowed values	Default / note
`featureControls.copy`	`allow`, `deny`	Required in the persisted policy bundle.
`featureControls.paste`	`allow`, `deny`	Required in the persisted policy bundle.
`featureControls.downloads`	`allow`, `deny`	Required in the persisted policy bundle.
`featureControls.uploads`	`allow`, `deny`	Required in the persisted policy bundle.
`featureControls.devtools`	`allow`, `deny`	Required in the persisted policy bundle.
`featureControls.audio`	`allow`, `deny`	Required in the persisted policy bundle.
`featureControls.automation`	`allow`, `deny`	Required in the persisted policy bundle.
`featureControls.contextMenu`	`allow`, `deny`	If omitted, BrowserBox currently defaults this field to `allow` for backward compatibility.

6.3.2 Enumerated policy fields

Field	Allowed values	Default / note
`transport.certificateErrors`	`enforce`, `ignore-all`	Defaults to `enforce`.
`navigation.mode`	`open`, `allowlist`, `blocklist`	Required in the persisted policy bundle.
`navigation.privateNetwork.mode`	`deny`, `allowlist`	Defaults to `deny`.

`featureControls.browserChrome`	`toggle`, `api`, `always-show`, `always-hide`	Defaults to `toggle`. Use `always-hide` for kiosks (§6.8).

6.3.3 Identity and access configuration

Three fields govern identity integration for authenticated BrowserBox deployments:

Field	Allowed values	Note
`identity.authMode`	`magic-link`, `sso-saml`	How users authenticate. `magic-link` uses email-based one-time links; `sso-saml` routes through an IdP.
`identity.ssoProvider`	`none`, `okta`, `entra`, `custom`	The IdP in use when `authMode=sso-saml`. Use `custom` for non-enumerated providers.
`identity.peerCapabilityModel`	`shared`, `scoped`	Whether multiple connected clients in the same session share capabilities (`shared`) or each client has its own scoped capability set (`scoped`).

6.3.4 List-valued policy fields

Field	Allowed entry form
`navigation.allowlist`	Array of non-empty strings. Entries may be exact hosts, exact origins, exact URLs, or wildcard host patterns; see §6.5.2 for how each matches.
`navigation.blocklist`	Array of non-empty strings. Entries follow the same matching forms as the navigation allowlist.
`navigation.internalAllowlist`	Array of exact, origin-only `chrome://page` or `chrome-extension://<32-character-id>` strings. Wildcards, paths, queries and fragments are rejected. An allowed extension origin covers paths within that exact extension.
`navigation.privateNetwork.allowlist`	Array of non-empty strings. Entries may be exact hosts, exact IPs, or IPv4 CIDRs.

6.3.5 Where customers can verify allowed values

Customers do not have to guess. BrowserBox exposes the current policy surface through the customer-facing policy CLI:

- `bbx policy show --scope global|user` prints one persisted policy source.
- `bbx policy show` prints the normalized effective policy that BrowserBox is actually using.
- `bbx policy validate --file policy.json` validates a candidate policy file against the current code and reports invalid enum values directly.

For example, if a customer uses an unsupported mode, BrowserBox validation reports the current accepted set, such as:

```
featureControls.browserChrome must be one of:
toggle, api, always-show, always-hide
```

6.4 Sensitive actions BrowserBox can allow or deny

These actions currently route through BrowserBox's policy authorization layer:

Action	Customer-visible effect
navigate	Page navigation, history navigation, new-tab target creation, and clean-slate navigation flows.
uploads	File upload routes and file chooser / file input flows.
downloads	Download and archive-serving routes. When denied, the download is cancelled before any data reaches the server's download directory, and the user sees a policy-denied notification in the BrowserBox client UI.
devtools	BrowserBox DevTools service and its websocket bridge.
audio	BrowserBox audio service.
copy	Keyboard and context-menu copy paths.
paste	Keyboard and API-driven paste / insert-text paths.
automation	The automation methods exposed through the webview API, such as selector waits, clicks, typing, and evaluation.
contextmenu	The remote-page context menu UI.

All of these controls are binary `allow` / `deny` decisions except `featureControls.browserChrome`, which uses explicit UI visibility modes described below.

6.5 Navigation, browser-internal and private-network policy

BrowserBox separates ordinary browsing, browser-internal access and private/local reachability:

- `navigation.allowlist` answers "where may the RBI tab browse?"
- `navigation.internalAllowlist` answers "which explicitly named browser-internal origins may open?"
- `navigation.privateNetwork.allowlist` answers "which private or local destinations may any request reach?"
- `ALLOWED_EMBEDDING_ORIGINS` is separate and only controls which parent pages may embed BrowserBox

Browser-internal navigation is denied by default, independently of `navigation.mode`. To permit an internal page, add its exact origin, for example `chrome://extensions`. To permit an extension's pages, add the exact extension origin, for example `chrome-extension://aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa`; BrowserBox then permits paths within that extension. `chrome://inspect`, `chrome://newtab`, `chrome://omnibox-popup`, and `devtools://` remain forbidden even if listed. If global and user policies are both active, both must list the origin. `about:blank` is structurally accepted so BrowserBox can stage safe tab creation, but a user-requested blank tab still follows the configured navigation mode.

The BrowserBox Extensions button is shown only when the effective internal allowlist contains `chrome://extensions` or a valid exact extension origin. This visibility check uses the policy snapshot only; it does not scan the installed extensions directory.

6.5.1 Navigation modes

Mode	Behavior
open	Any ordinary browsing target is allowed, subject to the separate private-network and browser-internal boundaries.
allowlist	Only configured allowlist entries may be reached.
blocklist	All targets are allowed except those matching the configured blocklist.

6.5.2 How allowlist entries match

Allowlist and blocklist entries take four forms. The form of the entry decides how it matches:

Entry	Form	Matches	Does not match
`*.example.com`	Wildcard host	`example.com` itself and every subdomain at any depth, such as `www.example.com` and `a.b.example.com`. Any scheme, port and path.	`notexample.com`, `example.org`
`www.example.com`	Exact host	`www.example.com` on any scheme, port and path.	`example.com`, `a.www.example.com`
`https://app.example.com`	Exact origin	Any path on that scheme, host and port. A trailing `/` makes no difference.	The same host over `http://`, or on port `8443`
`https://example.com/a?b=1`	Exact URL	That path and query only. A `#fragment` is ignored.	`/a`, `/a/c?b=1`, `/a?b=2`

Avoid these mistakes:

- **A path without a scheme allows the whole site.** An entry such as `example.com/maps` or `example.com/maps/*` is read as the host `example.com`. To allow only certain pages, list each page as an exact URL, scheme included. There are no path-prefix entries.
- **Only a leading `*.` is a wildcard.** An entry such as `*example.com` or `www.*.example.com` is read as a literal host name and matches nothing.

- **`bbx policy validate` does not flag either mistake.** Before you rely on an entry, test the addresses you mean to allow and refuse:

```
bbx policy check --action navigate --url https://www.example.com/
```

- **Every page load is checked**, including links, redirects and new windows opened from an allowed page. If a site sends users through another host, such as a sign-in or media host, that host needs its own entry.

  The page a user reaches when a navigation is denied can be replaced with your own; see §5.8.

6.5.3 Private-network controls

Private-network policy defaults to deny. Customer policy may widen that boundary explicitly:

Field	Meaning
mode=deny	Denies private and local targets such as RFC1918 addresses, localhost, and the unspecified addresses `0.0.0.0` and `::` by default.
mode=allowlist	Keeps the private/local boundary in place but permits explicit entries.
allowlist entries	Exact hosts, exact IPs, and IPv4 CIDRs, for example `10.0.0.5`, `10.0.0.0/8`, and `localhost`.

IPv4-mapped IPv6 addresses such as `::ffff:10.0.0.5` are classified as the IPv4 address they carry, so they cannot be used to bypass an IPv4 rule. An allowlist entry containing `/` must be a valid IPv4 CIDR; an invalid or IPv6 CIDR is ignored rather than treated as a host. Run `bbx policy validate` after editing the allowlist and `bbx policy check --action navigate --url <url>` to confirm the result for a specific destination.

6.6 Certificate-error policy

Certificate handling is now policy-driven rather than hidden behind a permanent launch flag.

Mode	Behavior
enforce	Default. BrowserBox enforces certificate validation and will not silently accept certificate errors.
ignore-all	Explicit opt-in for environments that need to reach self-signed or otherwise invalid HTTPS targets.

6.7 Browser chrome policy modes

`featureControls.browserChrome` does not allow or deny a risky action. Instead, it governs whether BrowserBox's own visible browser chrome, the tab bar and address bar, is shown, hidden, or user-toggleable.

Mode	Visible UI	User toggle	Meaning

toggle	visible by default	yes	Legacy-compatible mode. The login link's `ui=` sets the starting state. The user may show or hide BrowserBox chrome unless the embedder locks it.
api	visible by default	no	The embedder may drive UI visibility through the API, but the end user may not toggle it directly.
always-show	always visible	no	BrowserBox chrome must remain visible.
always-hide	always hidden	no	BrowserBox chrome stays hidden for kiosk-style or tightly embedded experiences. A `ui=true` login link, an edited address bar, the context menu and the embedder API cannot show it. For a kiosk, combine it with a navigation allowlist (§6.8).

If the embedder sets `ui-visible` or `allow-user-toggle-ui`, those values may further restrict the experience. They cannot widen what the server policy allows.

Navigation in always-hide mode: when chrome is hidden, BrowserBox's built-in back/forward buttons are not visible to the user. However, tab history is preserved internally. Embedders can navigate the remote browser's history programmatically through the webview API:

```
await webview.callApi('goBack');
await webview.callApi('goForward');
```

The remote-page context menu has no back option. Embedders that need end-user back navigation in always-hide mode should add their own back button wired to the `goBack` API call above.

6.8 Locking kiosk mode

A kiosk that users must not leave needs two settings in the server policy. The user's browser cannot override either of them.

- `featureControls.browserChrome` set to `always-hide`. The chrome stays hidden: a `ui=true` login link, an edited address bar, the context menu and the webview API cannot show it.
- A `navigation` allowlist naming the kiosk's sites (§6.5). The server refuses every page load outside it: typed addresses, new tabs, and links, redirects and new windows opened from inside an allowed site.

The allowlist is the boundary; `always-hide` removes the controls. Hiding the chrome alone is not a boundary, because a user who modifies the browser client could show it again. The allowlist still limits that user to the listed sites, so always set both.

Set them in the global policy, `/etc/browserbox/policy.json`. The global `browserChrome` setting takes precedence over a per-user policy, and the global policy applies to every BrowserBox Fleet seat on the host.

```
# If there is no global policy yet:
```

```
sudo bbx policy init --scope global --baseline regulated
sudo cp /etc/browserbox/policy.json kiosk.json
# In kiosk.json set:
#   "featureControls": { ..., "browserChrome": "always-hide" }
#   "navigation": { "mode": "allowlist", "allowlist": ["www.example.org"], ... }
sudo bbx policy set --scope global --file kiosk.json
```

New sessions start with the policy. In always-hide mode a user who follows a link off the allowlist has no back button, so install a `blocked-policy.html` page (§5.8.2) with a link back to the kiosk's start page.

6.9 Per-destination feature controls

`featureControls` settings such as `downloads` and `paste` are currently **global** — the same setting applies to all browsing destinations regardless of which site the user is on. Different `featureControls` per allowlist entry are not supported. If your use case requires different download or clipboard behavior on different sites, the recommended approach is to run separate BrowserBox instances each with a distinct policy configuration.

6.10 Server policy versus embedder policy

BrowserBox has three relevant policy layers:

1. **Global server policy**, stored in `/etc/browserbox/policy.json`, which applies to every Unix user and Fleet seat.
2. **User server policy**, stored in the user's existing BrowserBox policy directory, which may tighten but never loosen the global policy.
3. **Embedder policy hints**, supplied through the canonical webview surface, which may only restrict and never widen the effective server policy.

- An action is allowed only when every active server-policy scope allows it.
- With no global policy, the user policy behaves exactly as before.
- User and embedder restrictions may narrow capabilities further.
- An embedder cannot override a server-side deny into an allow.

6.11 Customer policy CLI

BrowserBox ships a policy CLI so customers can inspect, initialize, validate, and test the effective policy surface.

```
bbx policy help
bbx policy where [--scope global|user] [--json]
bbx policy baselines
bbx policy controls [--json]
bbx policy show [--scope global|user]
bbx policy get [--scope global|user]
bbx policy resolve
bbx policy check --action <name> [--url <https://...>] [--source <tag>]
bbx policy trace [--last <n>]
bbx policy init [--scope global|user] [--baseline regulated|compat]
```

```
bbx policy reset [--scope global|user] [--baseline regulated|compat]
bbx policy set [--scope global|user] --file <path/to/policy.json>
bbx policy validate [--scope global|user] [--file <path/to/policy.json>]
```

6.11.1 Useful commands

Command	Meaning
`bbx policy where`	Prints the resolved Unix principal plus global and user policy paths and status.
`bbx policy baselines`	Lists supported baseline names.
`bbx policy controls --json`	Prints the documented NIST 800-53 and FIPS 140 control surfaces.
`bbx policy show`	Prints the effective policy actually enforced, with source and principal diagnostics.
`bbx policy show --scope global`	Prints the raw stored global policy; use `--scope user` for the user's raw policy.
`bbx policy resolve`	Compatibility alias for the compiled / normalized effective policy snapshot.
`bbx policy check --action ...`	Evaluates a single action decision. Exit code `0` means allow; exit code `2` means deny.
`bbx policy trace --last 20`	Shows recent policy decisions from the decision trace log.
`bbx policy init --scope user --baseline compat`	Creates the selected scope from a named baseline.
`bbx policy reset --scope global --baseline regulated`	Replaces the selected scope with a named baseline.
`bbx policy set --scope user --file policy.json`	Loads, validates, and atomically persists a policy at the selected scope.
`bbx policy validate`	Validates the current persisted policy or an explicitly supplied file.

6.11.2 Recipe: global policy with a per-user tightening

Set the administrator-owned policy that applies to every BrowserBox Unix user and Fleet seat:

```
bbx policy set --scope global --file global-policy.json
```

Optionally set a stricter policy for one Unix user. The operator reads the source file and BrowserBox writes it in the target user's policy store, so the target account does not need access to the operator's file:

```
bbx policy set --scope user --file policy.json --for user-bbx-0901
```

Inspect the effective policy actually enforced for that user, then inspect the raw user layer if needed:

```
bbx policy show --for user-bbx-0901
bbx policy show --scope user --for user-bbx-0901
bbx policy where --for user-bbx-0901
```

The user layer may tighten the global policy but cannot weaken it. Reset only that user's layer to a named baseline with:

```
bbx policy reset --scope user --baseline compat --for user-bbx-0901
```

6.12 Operational visibility

When policy blocks an action, BrowserBox emits both customer-visible and operator-visible signals:

- a **policy-denied notification in the client UI** when a connected client is present at the time of the block
- server log entries prefixed with `[policy-decision]`
- a persistent trace file at `~/.config/dosaygo/bbpro/policy/decisions.ndjson`

Actions that currently produce client-visible notifications include:

Action	Notification trigger
navigate	A navigation to a blocked destination.
downloads	A browser download that was cancelled before any bytes reached disk.
copy, paste, devtools, uploads, automation, contextmenu	The corresponding action was attempted while denied by policy.
tab-limit	A new tab or popup was blocked because the session is at the configured tab cap (`BBX_MAX_TABS`).

Tab-limit blocks that occur during service startup are silent—they close excess tabs without showing a client notification, since no client is connected yet.

7 Embedding and Origin Allowlisting

For the hosted BrowserBox SaaS service, the current REST API and embed reference is published at https://hyper-frame.art/api/. That reference covers both session-lifecycle REST endpoints and the canonical `<hyper-frame>` custom element using the current method and event names.

7.1 Server-side embedding allowlist

For self-hosted BrowserBox embeds, the main server-side allowlist is:

```
export ALLOWED_EMBEDDING_ORIGINS="https://app.example.com https://*.customer.example:*"
```

This controls which parent pages may embed BrowserBox.
Supported patterns include:

- exact origins, for example `https://app.example.com`
- subdomain wildcards, for example `https://*.customer.example`
- trailing port wildcards, for example `https://app.example.com:*`
- both together, for example `https://*.customer.example:*`

Important boundary:

- `ALLOWED_EMBEDDING_ORIGINS` controls which parent pages may embed BrowserBox.
- It does not control where the remote browser may browse.
- Browsing targets are controlled by BrowserBox policy.

7.2 Canonical embedder controls

For new embeds, prefer explicit webview controls over encoding everything into the login URL:

- `embedder-origin="https://app.example.com"`
- `ui-visible="false"`
- `allow-user-toggle-ui="false"`
- `first-load-cleanse="https://example.com"` when the embedder should remove inherited tabs and open a known first page

  Minimal example:

```
<hyper-frame
  login-link="https://localhost:8888/login?token=..."
  embedder-origin="https://app.example.com"
  ui-visible="false"
  allow-user-toggle-ui="false"
></hyper-frame>
```

7.3 First-load tab cleansing

Some hosted or demo embedders need the first visible session state to be deterministic even when the remote browser already has tabs. The `first-load-cleanse` attribute runs once per `login-link`: it closes the current tab set when the webview API becomes ready, then optionally opens one replacement tab.

```
<hyper-frame
  login-link="https://localhost:8888/login?token=..."
  first-load-cleanse="https://duckduckgo.com"
></hyper-frame>
```

The value is intentional:

- a non-empty URL closes existing tabs, then opens that URL
- an empty value, such as `first-load-cleanse=""`, closes existing tabs and opens no replacement tab
- omitting the attribute leaves the existing session tabs alone

The public `firstLoadCleanse(url)` method has the same semantics for explicit embedder control. Passing a non-empty `url` opens that URL after the cleanse; passing an empty string performs the delete-only cleanse.

7.4 Embedder media permissions

Embedders that do not want BrowserBox to request microphone, camera, or display-capture access can set:

```
<hyper-frame
  login-link="https://localhost:8888/login?token=..."
  embedder-origin="https://app.example.com"
  media-permissions="none"
></hyper-frame>
```

The current values are:

Value	Meaning
`default`	BrowserBox uses its normal media behavior. This is also the fallback for missing, empty, or unknown values.
`none`	The embedder does not want BrowserBox to use microphone, camera, or display capture for this embedded session. The webview removes those features from the iframe `allow` attribute and syncs the setting into the BrowserBox client. Safari-family clients use this setting to skip the BrowserBox media explainer and the related preflight media request.

This is a coarse embedder media control. It is not a substitute for BrowserBox browsing policy, and it does not currently expose separate per-device settings such as audio-only or camera-only controls.

7.5 Embedder session unload warning

BrowserBox shows a `beforeunload` warning ("you are about to leave your remote browser") on the BrowserBox wrapper page itself when the end-user has interacted with the session. For embedders that own the outer page and do not want BrowserBox to interrupt the user's navigation of their own application, this can be suppressed:

```
<hyper-frame
  login-link="https://localhost:8888/login?token=..."
  embedder-origin="https://app.example.com"
  session-unload-warning="none"
></hyper-frame>
```

Value	Meaning
`default`	BrowserBox installs its wrapper-page `beforeunload` warning. This is also the fallback for missing, empty, or unknown values.
`none`	BrowserBox's wrapper-page `beforeunload` handler still runs, but is lazily evaluated at fire time: when the embedder's `session-unload-warning` is `none`, the handler refuses to block the unload. This design is race-free; the handler checks the current setting each time the user tries to navigate, so the embedder value does not need to win a setup-time race against the BrowserBox script load.

This setting does not affect the BrowserBox modal for `beforeunload` prompts originating from the remote page being browsed. Those are governed by the `beforeunload-behavior` attribute below.

7.6 Remote page beforeunload behavior

When a remote page being browsed inside BrowserBox attempts to block navigation via a `beforeunload` dialog, BrowserBox normally intercepts this and shows a modal to the user. For automated or programmatic use cases where navigation should never be blocked, this can be configured to automatically allow or block the navigation without showing a UI:

```
<hyper-frame
  login-link="..."
  beforeunload-behavior="leave"
></hyper-frame>
```

Value	Meaning
`default`	BrowserBox shows its standard modal, allowing the user (or the embedder via the Modal API) to decide.
`leave`	BrowserBox automatically clicks "Leave" (proceed with navigation) and does not show the modal UI.
`remain`	BrowserBox automatically clicks "Remain" (stay on page) and does not show the modal UI.

This design uses the same JiT-evaluation pattern as the session unload warning, making it robust against initialization races.

7.7 Modal lifecycle and dismissal API

BrowserBox modals are exposed to embedders using the same event convention as the rest of the webview surface: the BrowserBox frame sends a dashed message type with a data payload, and `<hyper-frame>` dispatches a `CustomEvent` whose `detail` is that payload. Where dotted aliases exist, they carry the same `detail` object.

The modal API should not require customers to learn separate methods for alert, confirm, prompt, file chooser, auth, or notice modals. Each modal is represented as data with an advertised list of actions. Customers either choose one of those actions or call the convenience dismissal method.

```
webview.addEventListener('modal-opened', event => {
  const modal = event.detail;
});

webview.addEventListener('modal-updated', event => {
  const modal = event.detail;
});

webview.addEventListener('modal-closed', event => {
  const result = event.detail;
});
```

For compatibility with the webview's existing alias style, the implementation may also emit dotted aliases:

- `modal.opened` for `modal-opened`
- `modal.updated` for `modal-updated`
- `modal.closed` for `modal-closed`

A modal detail object should be safe to expose to the embedding page:

```
{
  id: "modal-42",
  type: "notice",
  title: "Safari Permissions",
  message: "...",
  url: null,
  link: {
    title: "View Bug",
    href: "https://bugs.webkit.org/show_bug.cgi?id=189503",
    target: "_blank"
  },
  actions: [
    { id: "close", label: "OK", role: "close" }
  ],
  defaultAction: "close",
  dismissAction: "close",
  requiresInput: false
}
```

The public modal methods are:

```
const modal = await webview.getCurrentModal();

await webview.respondToModal({
  id: modal.id,
  action: modal.dismissAction
});

await webview.dismissModal();
```

The low-level API equivalents use the same method names:

```
await webview.callApi('getCurrentModal');
await webview.callApi('respondToModal', { id, action, value });
await webview.callApi('dismissModal', { id });
```

`dismissModal()` activates the modal's safest advertised dismissal action. The selection order is:

1. use the action whose `id` is `dismissAction`, when present;
2. otherwise prefer an action with role `cancel`;
3. otherwise prefer role `close`;
4. otherwise prefer role `deny`;
5. otherwise, if there is only one action, use that action;

6. otherwise reject with an ambiguous-action error.

Prompt-like modals can accept a value through `respondToModal`:

```
await webview.respondToModal({
  id: "modal-43",
  action: "ok",
  value: "customer supplied text"
});
```

Unsupported or sensitive operations are not required to be advertised as actions. `dismissModal()` still works whenever the modal has a cancel, close, deny, or single remaining action.

The webview policy capability names are `modals.read` for modal lifecycle events and `modals.respond` for `respondToModal()` and `dismissModal()`. Both are enabled by default for the canonical webview policy.

8 Setup and Backend Modes

8.1 `bbx setup`

Customer-facing setup shape:

```
bbx setup [--port|-p <port> | --free-port] [--hostname|-h <hostname>]
          [--token|-t <token>] [--zeta|-z]
          [--backend <http|https>]
          [--flipbook-record <dir>]
          [--flipbook-description <text>]
          [--for <user>|--for=<user>]
```

- `--port` chooses the main BrowserBox service port.
- `--free-port` picks a random main port between 10026 and 32765 whose five-port block (main port ±2) and Chrome port (main port −3000) are all free. It cannot be combined with `--port`. On Windows the option is `-FreePort` (`--free-port` also works), and ports Windows reserves, for example for Hyper-V, count as in use. Without either option, `bbx setup` reuses the saved port, or takes the lowest free block from 4026 upwards.
- `bbx setup` refuses a block that includes a port Chrome will not connect to (`ERR_UNSAFE_PORT`), such as 6665–6669, 5060 or 10080, whether the port was given with `--port` or chosen automatically.
- `--hostname` sets the hostname customers use.
- `--token` sets the login token explicitly instead of generating one.
- `--zeta` enables host-per-service mode.
- `--backend http|https` selects whether BrowserBox itself serves HTTP or HTTPS.
- `--flipbook-record` enables flipbook recording (see Section 19).
- `--flipbook-description` sets an optional description for the recording.
- `--for` runs setup on behalf of an unprivileged user (see Section 10). Both `--for <user>` and `--for=<user>` are accepted.

Runtime values are written to:

```
~/.config/dosaygo/bbpro/test.env
```

8.2 Login-token lifecycle

A normal BrowserBox setup has one login token for one configured BrowserBox runtime. Running `bbx setup` without `--token` generates a new random token; passing `--token` uses the supplied value. The selected token is stored in `test.env`. `bbx stop`, `bbx start`, and `bbx restart` do not rotate it: stop removes the published `login.link`, and the next start reconstructs the link from the retained runtime configuration. Re-running `bbx setup` without an explicit token rotates it.

The resulting login link is a persistent connector for that runtime, not a one-time-use link and not an end-user identity. It may be used again to reconnect while the runtime remains valid. Multiple clients that possess the same connector join the same shared browser state, subject to `BBX_MAX_CONNECTIONS`; this is BrowserBox's co-browsing model. Use a separate runtime or Fleet allocation wherever users require separate browser state. In Fleet deployments, applications integrate with `fleet acquire` and `fleet release` rather than invoking `bbx setup` directly for each external user.

8.3 Connection types keep their own configuration

Each way of launching BrowserBox — a plain `bbx start` (the `setup` configuration), `cf-run`, `zt-run`, `tor-run`, `ng-run`, and `win9x-run` — keeps its own saved configuration. When you switch from one to another, BrowserBox files the current `test.env` under the type that wrote it and loads the configuration of the type you are starting, printing a line such as `Loaded the setup configuration; cf had it last.` The saved copies live beside `test.env` as `test.env.profile.<type>`.

In practice this means a tunnel run no longer changes how a later plain `bbx start` behaves. For example, after `bbx cf-run` (which serves plain HTTP behind the tunnel), `bbx start` returns to the HTTPS address and login link from your last `bbx setup`.

Related behavior:

- `bbx restart` relaunches with the connection type BrowserBox was last started with, and prints which one it is using.
- `bbx start --port <p> --hostname <h>` saves a changed address: BrowserBox stops, re-runs setup with the new address (keeping the token, zeta mode, backend mode, and flipbook settings), then starts. Ports must be between 1024 and 65535.
- `user.env` (below) is never part of a saved configuration and always applies on top of whichever type is running.

8.4 Persistent configuration overrides (`user.env`)

Running `bbx setup` regenerates `test.env`, which overwrites any values you added there directly. To make overrides that survive a re-setup, place them in:

```
~/.config/dosaygo/bbpro/user.env
```

BrowserBox sources `user.env` after `test.env` every time it starts, so values set there take precedence and are never touched by `bbx setup`. The file is not created automatically—create it once and it persists indefinitely.

Example:

```
# ~/.config/dosaygo/bbpro/user.env
# Overrides that survive bbx setup regeneration.
export BBX_MAX_TABS=4
export BBX_SHUTDOWN_IDLE_MS=3600000
export BBX_CONNECTED_IDLE_MS=1800000
export BBX_CONNECTED_IDLE_WARNING_MS=60000
```

Any valid environment variable accepted by BrowserBox may be placed in `user.env`. The file is loaded by both `bbx start` (`bbx.sh`) and the service runner (`_conrun.sh`).

At service startup, the orchestrator exports every active `BBX_*` variable together with the standard proxy, CA, and sound-session variables described in Sections 3.7 and 3.8. The main, audio, docs, and DevTools services therefore receive the same relevant startup environment. If `user.env` contains proxy credentials, restrict it to the service user:

```
chmod 600 ~/.config/dosaygo/bbpro/user.env
```

8.4.1 Recommended high-throughput profile

The following profile is a good starting point for high-throughput deployments on well-provisioned hosts. Native BrowserBox keeps WebRTC enabled, measures it against WebSocket with sparse samples, and can select the consistently better path while retaining exact-frame fallback. Its capture profile is fixed at JPEG quality 80 and every frame; ACK pacing, bounded mailboxes, and frame coalescing keep slow consumers from creating an unbounded backlog. On more constrained or high-latency hosts, keep `BBX_LOW_END_MODE` at its default and validate settings against your workload.

```
# Avoid Chrome's forced low-end behavior on adequately provisioned hosts.
export BBX_LOW_END_MODE=false

# Optional when the deployment does not need BrowserBox audio.
export BBX_NO_AUDIO=true

# GPU hosts only; requires working vendor EGL/DRI userspace and permissions.
export BBX_GPU=true
```

Do not set `BBX_DISABLE_WEBRTC=true` in this profile. `BBX_GPU` is an explicit bare-metal hardware opt-in, not an automatic default, and it carries three prerequisites — all of them must hold before the flag is retained in production:

1. An actual hardware GPU is attached to the machine (for cloud instances, a GPU-equipped shape such as one with an NVIDIA L4 or T4).
2. The correct vendor drivers and EGL/DRI userspace are installed and the BrowserBox user has permission to access the device.
3. Chrome on that machine verifies acceleration is live: browse to `chrome://gpu` inside the BrowserBox session and confirm the Graphics Feature Status entries report green *"Hardware accelerated"*. If entries report "Software only" or SwiftShader appears as the renderer, the flag is not delivering hardware acceleration and should be removed.

Diagnostic settings such as `BBX_DIAG` and `BBX_CDP_METRICS` are intentionally omitted here because they are troubleshooting tools rather than production performance settings.

8.4.2 Cross-origin iframe instrumentation

Some pages embed third-party content in cross-origin iframes. Chrome may place those frames in separate renderer processes. BrowserBox now attaches to those targets by default so its supported in-frame capabilities remain available:

```
# Temporary compatibility fallback for a problematic site:
export BBX_OOPIF=false
bbx restart
```

Keep the default enabled unless diagnosing a specific compatibility issue. Disabling it removes BrowserBox instrumentation from separately rendered cross-site frames. See Section 5.4 for the persistent setting and support guidance.

8.5 Reverse proxy and HTTP backend

If BrowserBox sits behind nginx, Caddy, or another edge proxy that terminates TLS:

```
export BBX_EXTERNAL_TLS=true
bbx setup --hostname example.com --port 8888 --backend http
```

In HTTP backend mode the edge owns public DNS and TLS. BrowserBox does not run its direct-public-IP DNS gate or generate a BrowserBox certificate for that hostname. Internal and split-horizon names are resolved through the host's configured resolver rather than hard-coded public resolvers.

When installing a new host that will use this topology, export the external-TLS intent before invoking the installer so certificate acquisition is skipped during installation as well:

```
export BBX_EXTERNAL_TLS=true
curl -fsSL https://browserbox.io/install.sh | bash
bbx setup --hostname example.com --port 8888 --backend http
```

The older compatibility form is still available:

```
bbx setup --http-only
```

8.6 Reverse proxy: preserving service origins (audio and DevTools)

BrowserBox is a multi-service architecture. In addition to the main browser service, it runs separate services for audio streaming, document viewing, and remote DevTools. These services communicate over distinct ports and require their own origin (hostname + port combination) to function correctly.

When all services are proxied through a single origin, audio and DevTools will stop working. This is a browser security constraint: the audio service uses a protocol that requires a unique origin, and the DevTools bridge is likewise origin-sensitive.

8.6.1 The DNS facade approach (recommended)

The recommended pattern gives each service its own subdomain, all served over port 443. With a main port of `8888`, the four services and their facade hostnames are:

Service	Port	Derived from	Facade hostname
audio	8886	main − 2	`p8886.example.com`
docs	8887	main − 1	`p8887.example.com`
main	8888	main	`p8888.example.com`
devtools	8889	main + 1	`p8889.example.com`

When the first label of the hostname matches `p<port>`, BrowserBox derives the sibling service origins from it automatically and addresses them on port 443. This behaviour is driven entirely by the URL the client is served from, so it requires no special server mode: a standard `--backend http` setup behind your own proxy is sufficient.

Users must reach the main service through its facade hostname (https://p8888.example.com/login?token=...). A bare hostname without the `p<port>` label does not trigger the rewiring, and audio will not function.

Configure BrowserBox to serve plain HTTP behind the proxy, using the *parent* domain as the hostname:

```
export BBX_EXTERNAL_TLS=true
bbx setup --hostname example.com --port 8888 --backend http
bbx start
```

Then publish one virtual host per service. BrowserBox can generate the complete four-service configuration from the saved setup values:

```
# Inspect or send the configuration to an external edge operator.
bbx ng-config print > browserbox-facade.conf

# Check saved configuration and wildcard certificate coverage.
bbx ng-config validate

# Or install one BrowserBox-owned include, test nginx, and reload it.
bbx ng-config apply
```

`print` writes only nginx configuration to standard output and never changes the system. `validate` is also non-mutating. `apply` owns only a stable `<user>-browserbox-facade.conf` include; it writes atomically, runs `nginx -t`, and restores the prior BrowserBox include if validation or reload fails. It does not delete or rewrite operator-maintained nginx files.

The generated configuration contains one block in the following shape for each of the four ports:

```
server {
    listen 443 ssl;
    server_name p8886.example.com;      # repeat for p8887 / p8888 / p8889

    ssl_certificate     /path/fullchain.pem;
    ssl_certificate_key /path/privkey.pem;

    location / {
        proxy_pass http://127.0.0.1:8886;   # matching backend port
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-Proto https;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;

        # required: audio and DevTools use WebSockets
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
```

```
        proxy_set_header Connection "upgrade";

        # required: streaming must not be buffered or timed out
        proxy_buffering off;
        proxy_read_timeout 3600s;
    }
}
```

Requirements for this layout:

- DNS resolves each `p<port>.example.com` name to the proxy.
- The TLS certificate covers every facade hostname; a wildcard `*.example.com` satisfies this.
- The `Upgrade` and `Connection` headers are forwarded, and `proxy_buffering` is off. Without these the audio stream will not establish.
- If your facade hostnames are long, nginx may report `could not build server_names_hash`. Raise it in the `http` block: `server_names_hash_bucket_size 128;`

Where you prefer the existing all-in-one nginx workflow, see `bbx ng-run` (Section 9.2). Use `ng-run` when BrowserBox should own route generation and startup together; use `ng-config` when an operator or external edge owns nginx lifecycle.

8.6.2 Direct multi-port exposure (alternative)

If DNS subdomain routing is not available, expose each port directly from the edge without rewriting the request to a shared origin. Each port must be reachable as its own origin from the client browser.

8.6.3 Audio connects but is silent

If `bbx status` reports `Audio: Degraded`, or the client reports that audio is connected but nothing is heard, check the capture side in order:

1. **Is a sound server reachable for the service user?** Run `pactl info` as that user. If it fails, the user has no active session — see Section 3.8.
2. **Is the BrowserBox capture source present?** Run `pactl list short sources` as the service user and confirm that `rtp.monitor` is listed. On a PipeWire host, if only `auto_null.monitor` appears, run `browserbox --install` to reapply the command-install assets, then stop and restart the BrowserBox runtime so `pipewire-pulse` reloads its configuration.
3. **Is the browser feeding it?** With a page playing sound, run `pactl list short sink-inputs`. An active stream should be listed. On PipeWire, the stream should use the `channel` sink. If the list is empty, the browser is not rendering into the sound server.
4. **Was the browser started inside the service user's session?** A browser process started outside that session cannot reach the sound server and will render to a null output. Run `bbx stop`; it terminates Chrome roots owned by the BrowserBox profile while leaving unrelated personal Chrome processes untouched. Then start BrowserBox again from the service user's session.

8.6.4 Minimal mode (no audio or DevTools)

For environments where audio and DevTools are not needed, run in minimal mode to avoid starting those services entirely:

```
export BBX_MINIMAL_MODE=true
bbx start
```

8.7 External TLS hint

If BrowserBox is behind an edge that terminates HTTPS and BrowserBox itself is serving plain HTTP, the runtime can be told to treat the frontend as secure:

```
export BBX_EXTERNAL_TLS=true
bbx start
```

This is useful when a reverse proxy or cloud edge is doing TLS termination for customer traffic.

9 Tunnel and Network-Oriented Commands

9.1 `bbx ng-config`

`ng-config` is the operator-controlled nginx surface for a BrowserBox instance that has already been configured. It reuses the saved hostname, main port, backend mode, service-host mappings, and certificate directory:

```
bbx ng-config print
bbx ng-config validate
bbx ng-config apply
```

Action	Behavior
print	Emits a complete nginx configuration to standard output. It does not install nginx, write files, validate certificates, or reload services. This is the appropriate action when nginx runs on another host or changes pass through operator review.
validate	Confirms the saved BrowserBox port/domain configuration and checks that `fullchain.pem` and `privkey.pem` are readable and that the certificate covers all four generated service hostnames. It does not change nginx.
apply	Installs or replaces the single BrowserBox-owned facade include, runs `nginx -t`, and reloads or starts nginx. A failed syntax check or reload restores the prior BrowserBox include.

With main port `8888` and parent domain `example.com`, the default output maps `p8886` through `p8889` to the audio, docs, main, and DevTools ports described in Section 8.6. When zeta/host-per-service configuration already exists, `ng-config` reuses those persisted `ADDR_<port>` names rather than creating a second hostname scheme.

For an external nginx host:

```
# BrowserBox host
bbx ng-config print > /tmp/browserbox-facade.conf

# Review and transfer through your normal configuration process.
scp /tmp/browserbox-facade.conf edge-admin@edge.example:/tmp/
```

The generated certificate paths come from the BrowserBox host's saved certificate directory: `BBX_SSLCERTS_DIR`, or `SSLCERTS_DIR`. If the edge stores the certificate elsewhere, change only those two paths during operator review; the service hostname and port mappings remain authoritative.

For nginx on the BrowserBox host:

```
bbx ng-config validate
bbx ng-config apply
```

9.2 bbx ng-run

```
bbx ng-run
```

This is the nginx-oriented workflow. It will trigger zeta-mode setup when needed, expects `hosts.env` support for per-service hostnames, runs nginx setup, and then starts BrowserBox.

Route domain selection. `bbx ng-run` creates per-service hostnames under a route domain. Because of that, `localhost` is not a valid `ng-run` route domain: names such as `audio.localhost` or `devtools.localhost` are not portable DNS targets and should not be relied on. `localhost` remains valid for the direct `bbx run` and `bbx start` workflows.

For local nginx testing, use a wildcard-safe local name such as `bbx.test` or `ci.test`:

```
bbx setup --hostname bbx.test --port 11111 -z
bbx ng-run
```

`ng-run` chooses its route domain in this order:

1. An explicit caller override, such as `DOMAIN=example.test bbx ng-run`.
2. The saved `DOMAIN` from the previous `bbx setup`.
3. The saved setup hostname, when it is suitable for generated subdomains.

Use `DOMAIN` only when you need to override the saved setup domain for this run:

```
DOMAIN=example.test bbx ng-run
```

If the final resolved route domain is `localhost`, BrowserBox refuses to continue and asks for a wildcard-safe domain instead. This protects nginx routing and certificate generation from creating unusable subdomain URLs.

9.3 bbx tor-run

```
bbx tor-run [--anonymize|--clearnet-only|--no-darkweb] [--onion|--no-onion]
```

This mode sets up Tor integration and can expose BrowserBox through onion routing. `--anonymize` routes the remote browser's own traffic through Tor; `--clearnet-only` (also spelled `--no-darkweb`) keeps browsing on the normal internet. `--onion` publishes BrowserBox as an onion service. At least one of anonymized browsing or the onion service must be enabled.

9.4 bbx zt-run

```
bbx zt-run [--network-id <16 hex characters>]
```

This is the ZeroTier overlay-network entry point for private-network access. In an interactive terminal, `bbx` prompts for the ZeroTier network ID when `--network-id` is omitted. In scripts and other non-interactive use, pass `--network-id`; without it the command exits with `ZeroTier Network ID is required.`

9.5 bbx cf-run

```
bbx cf-run [--port|-p <port>] [--background|-d]
```

This is the Cloudflare Tunnel entry point for public HTTPS exposure through Cloudflare's edge. The tunnel's public login link is written to `~/.config/dosaygo/bbpro/login.link` as soon as Cloudflare issues it, rewritten if the tunnel address changes, and removed when the tunnel stops. Scripts can therefore read the current link from that file instead of parsing console output.

10 Cross-User Execution (--for)

The `--for` flag lets a privileged operator run BrowserBox commands on behalf of an unprivileged runtime user. This is the recommended deployment pattern for multi-tenant or security-hardened environments where the runtime user should have no `sudo` capability.

10.1 Requirements

- The *operator* must have passwordless `sudo` (`sudo -n`).
- The *target user* must already exist and must **not** have `sudo`.
- Linux only — macOS does not support `--for`.

10.2 Supported commands

```
bbx setup   --for <user> [setup-options]
bbx run     --for <user>
bbx ng-run  --for <user>
bbx stop    --for <user>
bbx status  --for <user>
bbx policy  <subcommand> --for <user>
```

`start`/`run` and `ng-start`/`ng-run` are equivalent spellings. Both `--for <user>` and `--for=<user>` syntax are accepted.

10.3 What happens

1. **Operator-level tasks** (TLS certificate generation, nginx configuration, host file entries, port binding) run as the operator via `sudo`.
2. **Runtime tasks** (BrowserBox services, license certification, browser process) run as the target user via `sudo -u <user>`.

3. SSL certificates generated during setup are automatically `chown`ed to the target user so that the BrowserBox runtime can read them.
4. Configuration is stored under the target user's home directory (`~/<user>/.config/dosaygo/bbpro/`).

10.4 Example workflow

```
# As the operator (e.g. bbxadmin, with sudo):
bbx setup --port 9090 --hostname app.example.com -z --for bbxruntime
bbx ng-run --for bbxruntime

# Later:
bbx status --for bbxruntime
bbx stop   --for bbxruntime
```

All BrowserBox processes will run as `bbxruntime`. The operator never needs to `su` or log in as the runtime user.

10.5 Combining with certificate override

`--for` respects `BBX_SSLCERTS_DIR` when set explicitly:

```
export BBX_SSLCERTS_DIR="/opt/shared-certs"
bbx ng-run --for bbxruntime
```

If `BBX_SSLCERTS_DIR` is not set, certificates are generated into the target user's default location (`~/<user>/sslcerts/`).

11 Zeta Mode and `hosts.env`

The `--zeta` option enables host-per-service mode. BrowserBox then expects:

```
~/.config/dosaygo/bbpro/hosts.env
```

At runtime, `bbx run` reads `ADDR_<PORT>` from that file to construct the login link for the main service port.

Use zeta mode when:

- nginx is doing hostname-based proxying
- you want separate hostnames per BrowserBox service
- you want clean standard-port edge routing

If `hosts.env` is missing, `bbx run` fails fast in zeta mode.

12 Fleet: A Pool of Ephemeral Sessions (`bbx fleet`)

Fleet turns a sufficiently large Linux machine into a pool of ephemeral, clean-slate BrowserBox sessions. It is the machine-level deployment mode for applications that need to hand each authenticated end user a fresh, disposable browser session — without creating static BrowserBox tokens per user, and without your application ever managing Linux users, ports, nginx, or browser profiles.

Fleet is **Linux only** and requires a privileged operator (root, or a non-root account with passwordless `sudo`). Fleet seat users themselves never receive `sudo`.

12.1 The mental model

Do *not* mint static BrowserBox tokens for every end user of your application. Instead, your application allocates a session at the moment it is needed:

```
application authentication succeeds
        |
bbx fleet acquire --json --json-schema 2
        |
redirect user to login_url
        |
session closes or expires
        |
bbx fleet release <allocation_id>
```

The login token inside `login_url` is a fresh, allocation-scoped connector capability — it is not your application's user identity and is not consumed on first use. Your application remains responsible for its own authentication (Turnstile/CAPTCHA, SAML or OIDC, JWT validation, authorization), for its own session-timeout and disconnect detection, and for calling `fleet release` when a session ends. Fleet's runtime monitor is a recovery path for BrowserBox processes that exit independently; it does not replace application-level release. Fleet handles everything below that line: Linux seat users, BrowserBox setup, allocation-scoped tokens, ports, nginx routing, startup, shutdown, clean-slate profile lifecycle, and local allocation locking.

12.2 The three layers

- **Fleet deployment** — machine-level configuration: the seat pool, port range, public hostname and routing topology, wildcard TLS, fleet-wide BrowserBox defaults, clean-slate policy.
- **Fleet seat** — a reusable OS identity (e.g. `bbx-seat-0000`) that can run exactly one allocation at a time. Seats are an implementation detail; integrate against allocation IDs, not usernames.
- **Fleet allocation** — one ephemeral session: a reserved seat, a reserved port set, a fresh login token, a clean browser profile, one public login URL, and one opaque allocation ID.

12.3 Connector and co-browsing semantics

The Fleet allocation is the isolation boundary. Its `login_url` may be used repeatedly for reconnects while that allocation is alive. If more than one client receives the same URL, those clients join the same remote browser and see and control the same tabs, cookies, and page state, up to the configured `BBX_MAX_CONNECTIONS` cap. This is intentional co-browsing, not the creation of another isolated session and not the consumption of another Fleet seat.

Treat `login_url` as a bearer secret: possession is sufficient to join the allocation. Do not distribute one allocation's URL to unrelated users. For one isolated member session per user, acquire once per member session and deliver that URL only to that authorized client. Release or automatic recovery stops the runtime and removes its published login capability; a later acquisition, even on the same reusable seat, receives a different random token.

12.4 Requirements

- Linux, with root or passwordless `sudo` for the operator.

- An installed BrowserBox and a license with enough seats for your target concurrency, configured for the privileged operator that runs Fleet.
- Machine sizing appropriate to concurrency (each live session runs a real Chrome).
- For production subdomain routing: wildcard DNS (`*.your-domain`), a wildcard TLS certificate, nginx, and public port 443.
- An internal port range for BrowserBox backends (default 7000–20000, loopback only in subdomain mode).
- On PipeWire hosts, BrowserBox's system `pipewire-pulse` drop-in. This is installed automatically by `bbx update` and full installation; `fleet init` and `fleet doctor` verify it before sessions are allocated.

12.5 Privileged operator and licence

Fleet deployment state and licence configuration belong to the operating-system identity that runs `bbx fleet`. In the common `sudo bbx fleet ...` arrangement, that identity is `root`. A key previously certified only as the unprivileged `browserbox` user is intentionally not read from that user's private configuration.

Configure the key once in the privileged operator context before the first acquisition, then confirm the doctor check passes:

```
sudo -H bbx certify
sudo bbx fleet doctor
```

If the privileged command must use a forward proxy, pass the standard proxy environment while certifying. The key is still entered at the interactive prompt and need not appear in shell history:

```
sudo -H env \
  http_proxy="$http_proxy" https_proxy="$https_proxy" \
  HTTP_PROXY="$HTTP_PROXY" HTTPS_PROXY="$HTTPS_PROXY" \
  no_proxy="$no_proxy" NO_PROXY="$NO_PROXY" \
  bbx certify
```

12.6 Initialization

```
sudo bbx fleet init \
  --size 100 \
  --domain browser.example.com \
  --routing subdomain \
  --subdomain-mode random \
  --cert-file /etc/browserbox/fullchain.pem \
  --key-file /etc/browserbox/privkey.pem
```

Initialization is idempotent: re-running it creates only missing seat users, increasing `--size` adds seats, and decreasing `--size` never deletes users (excess seats simply become ineligible for new allocations). The nginx configuration is replaced atomically after certificate and DNS validation.

Fleet enables systemd lingering for every new or existing managed seat user so PipeWire or PulseAudio remains available on a headless host. On PipeWire systems, initialization also refuses to complete unless the BrowserBox system audio drop-in is installed. If this check fails after an

upgrade, run `browserbox --install` (the same command-install step normally performed by `bbx update`), then repeat the original `fleet init` command. A full machine installation is not required.

A successful subdomain initialization installs and reloads the complete Fleet nginx configuration; a separate `fleet routing apply` is not required. For automation, add `--json`. Both success and failure then produce one structured result on standard output while operational detail remains on standard error.

If a first initialization stops after creating users but before routing is complete, the users and stable seat records are retained for the retry. This is safe preparatory state, not an allocatable Fleet: `fleet doctor` reports `Fleet deployment ready for acquire` as failed, and `fleet acquire` returns `fleet_not_ready`. Correct the reported prerequisite and re-run the complete `fleet init` command with the same routing and certificate options.

Each seat receives a *stable* main port and, in subdomain mode, *stable* public hostnames, so nginx is configured completely at init time — `acquire` and `release` never touch DNS, certificates, or nginx.

For conventional browsers, Fleet's distinct service hostnames retain the normal iframe-authenticated WebSocket/WebAudio path; the service-name map carried by the login flow supplies the otherwise unrelated main and audio origins. Tor mode retains its direct WAV transport. Applications should therefore use the returned `login_url` unchanged rather than reconstructing service URLs.

12.7 DNS

Subdomain routing requires a wildcard record pointing at the machine:

```
Type: A
Name: *.browser.example.com
Value: 203.0.113.10
```

Add a matching AAAA record if the machine serves IPv6. `fleet init` prints the exact records required, then validates two distinct probe subdomains before touching nginx. If your traffic deliberately arrives through a load balancer or proxy (so DNS does not resolve directly to this machine), pass `--allow-proxied-domain`; Fleet then distinguishes "DNS resolves successfully" from "DNS is confirmed to resolve directly to this machine" rather than silently waiving validation.

12.8 Certificates

Subdomain routing needs a certificate covering `*.your-domain` (and ideally the base domain). Supply an existing wildcard certificate with `--cert-file`/`--key-file`; Fleet validates existence, key permissions, PEM type, cert/key match, expiry, and wildcard SAN coverage. Because wildcard issuance requires DNS validation, Fleet does not attempt automated wildcard issuance; issue the certificate through your DNS provider (for example `certbot` DNS-01) and hand the files to `fleet init`. Local/test domains (`*.test`, `*.localhost`, etc.) get locally generated certificates automatically.

The key must be the TLS private key belonging to the certificate, not an SSH key. Its first line is normally `BEGIN PRIVATE KEY`, `BEGIN RSA PRIVATE KEY`, or `BEGIN EC PRIVATE KEY`; `BEGIN OPENSSH PRIVATE KEY` is not accepted by nginx. Keep the key owner-only or group-read-only (normally mode 600 or 640), then verify the material before initialization:

```
sudo chmod 600 /etc/browserbox/privkey.pem
sudo openssl pkey -in /etc/browserbox/privkey.pem -noout -check
```

```
sudo openssl x509 -in /etc/browserbox/fullchain.pem \
  -pubkey -noout | openssl sha256
sudo openssl pkey -in /etc/browserbox/privkey.pem \
  -pubout | openssl sha256
```

The two SHA-256 values must match. Fleet reports distinct errors for an unreadable X.509 certificate, an OpenSSH or otherwise invalid TLS key, unsafe key permissions, a certificate/key mismatch, expiry, and missing wildcard SAN coverage.

The supplied paths remain the certificate source of record. Fleet installs a root-owned copy for nginx under its system configuration tree, with restrictive modes and the platform's normal security context. This permits a source certificate to remain under an operator-controlled home or application directory on SELinux-enforcing RHEL-family systems. Fleet also applies the standard SELinux reverse-proxy allowance when required so nginx can reach the loopback BrowserBox backends.

After renewing or replacing the source files in place, refresh nginx atomically with:

```
sudo bbx fleet routing apply
```

12.9 Forward-proxy defaults for Fleet seats

Fleet seats are separate operating-system users. Persist proxy variables through Fleet's existing defaults surface so delegated setup, licence certification, BrowserBox services, Chrome, and restart guardians receive the same network policy on every acquisition:

```
sudo bbx fleet config set http_proxy  http://proxy.example.com:3128
sudo bbx fleet config set https_proxy http://proxy.example.com:3128
sudo bbx fleet config set HTTP_PROXY  http://proxy.example.com:3128
sudo bbx fleet config set HTTPS_PROXY http://proxy.example.com:3128

sudo bbx fleet config set no_proxy \
  localhost,127.0.0.1,::1,10.0.0.0/8,.internal.example.com
sudo bbx fleet config set NO_PROXY \
  localhost,127.0.0.1,::1,10.0.0.0/8,.internal.example.com
```

These values are stored in the operator-private Fleet state and are passed through the same delegated environment path as other Fleet-wide BrowserBox defaults. Use `sudo bbx fleet config show` to review them. This is distinct from the privileged operator environment used by `fleet init`, `fleet doctor`, and `bbx certify`; provide proxy variables to that operator process as shown above when it also requires proxy egress.

12.10 Routing modes

- **`subdomain`** (production default): every session is presented through port 443 under a per-seat hostname. Label styles: `port` (`p7000.domain`), `seat` (`seat-0000.domain`), or `random` (`k7p4wm9f.domain` — recommended: stable opaque hostnames, clean URLs). The hostname is stable per seat across allocations; the login token is fresh for every acquisition.
- **`direct-port`**: the returned URL uses the machine hostname and the seat's port directly (`https://host:7000/login?token=...`). Suitable for testing, internal deployments, or environments with another upstream routing layer. May require opening the port range in your firewall.

Routing is configured fleet-wide, not per acquisition: this keeps the session-start path fast (no DNS change, no certificate operation, no nginx reload) and keeps public URLs stable and auditable. Changing routing topology is a deployment-wide operation — `bbx fleet routing apply` refuses to run while allocations are active unless you explicitly pass `--allow-active`.

12.11 Acquire and release

```
session_json="$(sudo -n bbx fleet acquire --json --json-schema 2 2>/dev/null)"

allocation_id="$(jq -r '.allocation.id' <<<"$session_json")"
login_url="$(jq -r '.allocation.login_url' <<<"$session_json")"

# Redirect the already-authenticated application user to $login_url.

sudo -n bbx fleet release "$allocation_id"
```

Schema 2 is the recommended integration contract. Acquire and single-allocation status return `{ok, schemaVersion, allocation}`, while list returns `{ok, schemaVersion, allocations}`. Every allocation uses the same fixed field set: `id`, `state`, `health`, `seat`, `main_port`, `public_hostname`, `login_url`, `created_at`, `updated_at`, and `timeout_seconds`; unavailable values are JSON `null`. Status-only probes live under `allocation.diagnostics`. Schema 1 remains the default during the compatibility window and can be selected explicitly with `--json-schema 1`.

In `--json` mode, stdout carries a single JSON document while diagnostics go to stderr. A terminal displays both streams together, which can make `[fleet]` progress lines appear to precede the JSON even though stdout remains valid. Capture stderr separately in production when its diagnostics matter; redirecting it as above is appropriate when only the result object is required. Errors carry stable codes such as `fleet_not_ready`, `fleet_exhausted`, `fleet_license_unavailable`, or `fleet_public_route_failed`. Acquisition succeeds only after the public login URL is verified live; any failure (license, setup, startup, routing) rolls the local reservation back.

The privileged operator licence is checked before a seat is reserved. If delegated BrowserBox setup itself fails, Fleet writes a bounded, secret-redacted diagnostic to `fleet/diagnostics/<seat>-setup.log` and names that path in the error response.

12.12 Keeping a small warm pool

Fleet does not add a second "allocated but unassigned" state. A customer application can provide sub-second handout by acquiring a small number of normal allocations ahead of demand and recording them as warm in its own database. Size the deployment so that Fleet seats cover peak assigned sessions, warm stock, and operational headroom:

```
fleet size >= peak assigned + warm target + headroom
```

Each warm allocation consumes a Fleet seat and a licence seat. Its `login_url` is also a live bearer capability, so store it with the same care as any other session credential and never write it to application logs.

The idle timer makes warm stock bounded. `BBX_SHUTDOWN_IDLE_MS` starts when the BrowserBox runtime starts, before any client has connected. The first client connection cancels that timer; the timer starts again after the last client disconnects. A user who receives a healthy warm allocation therefore gets a fresh idle budget rather than the remainder of its pool dwell time.

Configure a stock lifetime comfortably longer than the expected dwell time and keep the existing Fleet monitor running as a recovery path:

```
# Example: allow warm stock to dwell for up to two hours.
sudo bbx fleet config set BBX_SHUTDOWN_IDLE_MS 7200000

# Run under systemd or the deployment's normal supervisor.
sudo bbx fleet monitor --interval 5 --grace 15
```

The application owns four bounded loops:

1. **Top up.** Every few seconds, and immediately after a handout, acquire only the number needed to restore the warm target. Keep acquire concurrency small because acquisitions briefly share Fleet's local lock.
2. **Hand out.** Pop the oldest warm row, mark it assigned, and return its `login_url`. Before returning it, call `bbx fleet status <allocation-id> --json --json-schema 2` and require `allocation.health=healthy`, `allocation.diagnostics.login_link_present=true`, and `allocation.diagnostics.public_route_ok=true`. Bound retries and fall back to a cold acquire if no healthy warm row remains.
3. **Rotate.** Release and replace warm rows after roughly 75 percent of the configured idle budget so a user is not handed a session near shutdown.
4. **Return.** On explicit logout, application timeout, or another terminal session event, call `fleet release`, remove the row, and trigger top-up.

Fleet health values are `healthy`, `partial`, `down`, and `unknown`; there is no `up` value. Reconcile the application's rows against `bbx fleet list --json --json-schema 2` before top-up in both directions: drop rows Fleet no longer has, and adopt or release live allocations left behind by an interrupted application write. This prevents dead handouts, under-provisioning against phantom rows, and stranded paid seats.

In a measured Rocky Linux 9 proof with SELinux enforcing, three cold acquires averaged 4.260 seconds while three warm handouts, including the health pre-flight, averaged 0.699 seconds. These values demonstrate the regimen rather than promise a universal latency; host CPU, storage, Chrome profile creation, and Fleet size affect cold acquisition time.

12.13 Calling Fleet from a web application

No additional BrowserBox HTTP provisioning API is required. A web application on the Fleet host may execute the fixed argument vectors `bbx fleet acquire --json --json-schema 2` and `bbx fleet release <allocation-id> --json --json-schema 2` directly through a small privileged adapter. A remote application may call the same adapter over a local authenticated service boundary. Keep the adapter narrow:

- expose only acquire, release, and optional status operations;
- invoke `bbx` with an argument array, never a shell command assembled from request text;
- authenticate and authorize in the application before acquisition;
- validate allocation IDs before release and never accept a seat username, port, command, or file path from the request;
- return `login_url` to the authorized caller without logging it; outside a warm-pool record, retain only `allocation_id`;

- release on explicit logout and application timeout, with Fleet monitor/reap retained as recovery rather than the normal application lifecycle.

Generic fixed-hook runners such as `adnanh/webhook` can provide this transport when they are already part of the deployment. Bind the runner to loopback or a protected internal interface, require application authentication or mTLS at the edge, configure fixed hook commands with no request-derived shell arguments, and give it only the privilege needed for the exact Fleet commands. Do not expose a general command runner or the raw privileged `bbx` CLI to the network.

12.14 Clean-slate security

Every allocation starts from a fresh browser profile by default (`FLEET_CLEAN_SLATE=true`), using BrowserBox's canonical `BBX_CLEAN_SLATE` mechanism. Cookies, local storage, history, cache, saved authentication, IndexedDB, and service-worker state from a prior user of the same seat are not intended to survive seat reuse: release stops the runtime, removes the login capability, and wipes the browser profile; acquisition additionally starts with clean-slate enforced. Because different external users receive the same reusable seats over time, do not disable clean slate in multi-user deployments. Persistent per-user profiles are deliberately not part of this design.

12.15 Operational commands

```
sudo bbx fleet list              # active allocations (no secrets)
sudo bbx fleet status [<id>]     # fleet summary or one allocation
sudo bbx fleet reap              # one dead-runtime recovery pass
sudo bbx fleet monitor           # continuous foreground recovery
sudo bbx fleet doctor            # environment / DNS / cert health
sudo bbx fleet reconcile --fix   # repair state drift safely
sudo bbx fleet routing show      # seat-to-hostname-to-port map
sudo bbx fleet routing apply     # atomic whole-of-fleet re-apply
sudo bbx fleet config set K V    # fleet-wide BrowserBox defaults
```

`fleet status` reports local capacity, fully down and partially healthy running allocations, and an *advisory* license-vacancy snapshot; local free seats do not guarantee globally free license seats — license certification during acquire remains authoritative.

12.16 Automatic dead-runtime recovery

BrowserBox can shut itself down when a runtime has never received a client connection or after its last client disconnects; the main timeout is controlled by `BBX_SHUTDOWN_IDLE_MS`. The timer starts at runtime startup, is cancelled by the first client connection, and starts again after the last client disconnects. Native BrowserBox can additionally enforce opt-in connected-client inactivity with `BBX_CONNECTED_IDLE_MS`, as described above. Either shutdown path uses the runtime's normal graceful close. Fleet confirms the runtime is fully down and then returns the allocation through the same stop, login-capability removal, and clean-slate profile-wipe path used by an explicit `fleet release`. A customer application must still enforce any absolute-session limit and call `fleet release` on explicit logout.

Run the foreground monitor under your normal service supervisor:

```
# Persist the BrowserBox idle timeout for every Fleet seat (15 minutes).
sudo bbx fleet config set BBX_SHUTDOWN_IDLE_MS 900000
```

```
# Optional: reclaim a connected seat after 30 inactive minutes.
sudo bbx fleet config set BBX_CONNECTED_IDLE_MS 1800000
sudo bbx fleet config set BBX_CONNECTED_IDLE_WARNING_MS 60000

# Run this foreground process under systemd or your supervisor.
sudo bbx fleet monitor
# Optional:
sudo bbx fleet monitor --interval 5 --grace 15
```

A running allocation is eligible for automatic release only when its main port is no longer listening and the seat has no live BrowserBox or Chrome process. Zombie process-table entries do not count as a live runtime. A partial result (only the port or only a live process remains), or an unknown result caused by a failed process-table probe, is reported by `fleet list` and `fleet status`, but is not automatically released. The monitor observes process and socket health outside the short Fleet ownership lock, waits through the grace window, and then revalidates the allocation identity, state, runtime marker, and health before cleanup. A recovered or replaced runtime is therefore left alone, and a slow health probe does not block a simultaneous acquire. Stopping the monitor during that window cancels the in-progress pass rather than shortening the grace.

`fleet reap` performs one such pass. On its normal path, `fleet acquire` makes one locked pass over the seat and allocation records, reserves one seat atomically, and performs startup outside the lock; it does not scan the process table or wait for monitor work. If the records say the pool is exhausted, acquire performs one bounded recovery pass using the shorter acquire grace, retries reservation once, and then reports exhaustion. This just-in-time fallback recovers capacity if the monitor is temporarily unavailable while keeping the ordinary acquire path lightweight. Two simultaneous acquires cannot reserve the same seat or overlapping port set, and simultaneous releases are idempotent.

The monitor is quiet when no recovery candidates exist. A pass with candidates emits one summary rather than one success line per allocation. Failure detail is capped (10 allocations per pass by default), additional detail is summarized as suppressed, and repeated failed passes use exponential backoff capped at 300 seconds. Stale `reserved`, `starting`, or `releasing` records are owned by monitor/reap recovery after their transition TTL (900 seconds by default); acquire does not perform heavy stale-state repair on its normal path.

12.17 Security guidance

- `login_url` is a secret capability. Do not log it. Deliver it only to the authenticated user and then forget it; a warm-pool row is the exception and must protect the URL as a live credential until handout or release. Retain `allocation_id` for lifecycle operations.
- The login URL is reusable for the life of its allocation and may admit co-browsers. Never describe it as a one-time-use credential or share it across users who require isolation.
- Call `fleet release` on session timeout *and* explicit logout.
- Fleet seat users must never have `sudo`.
- Fleet state (`~/.config/dosaygo/bbpro/fleet/`) is operator-private (0700/0600).
- Human-readable `fleet list` output never shows tokens; `--json` output (which does include `login_url`) is for the privileged operator only.

12.18 Reference architecture

A typical hardened deployment in front of Fleet:

```
Cloudflare Tunnel / reverse proxy
        |
Turnstile
        |
SAML identity provider
        |
application authorization and JWT
        |
bbx fleet acquire          (this machine)
        |
BrowserBox public Fleet hostname (port 443)
        |
target pension/member portal
```

The SAML/JWT system and Fleet are separate layers: your application proves who the user is; Fleet hands that user one disposable browser session and takes it back afterwards.

13 Docker Deployments (`bbx docker`)

Each BrowserBox release publishes a container image for Linux on AMD64 and ARM64. The image holds the native BrowserBox binary, Chromium and the normal `bbx` command layer, so a container host needs neither a host Chromium nor a host BrowserBox installation. The same images are published to two registries:

```
ghcr.io/browserbox/browserbox:20.4.0    # or :latest
docker.io/dosyago/browserbox:20.4.0     # or :latest
```

Docker pulls the variant for the engine's own platform. `bbx docker` manages a named deployment of this image on a Docker engine. The image does not depend on it: it also runs directly on your own container infrastructure (Section 13.7).

13.1 Quick start

A full BrowserBox installation already includes the Docker commands. A host that should only run containers can install the commands alone:

```
curl -fsSL https://browserbox.io/install.sh -o install.sh
bash install.sh --cli-only
```

The commands go to `/usr/local/bin` when it is writable (directly or through sudo), otherwise to `~/.local/bin`; `BBX_CLI_BIN_DIR` chooses another directory. Then create and start a deployment named `work`:

```
bbx docker run work
```

Deployment names contain 1–59 lowercase letters, digits or hyphens and begin with a letter or digit. Together with the `bbx-` prefix, this keeps the generated Compose project name within one portable 63-character label.

The first run configures the deployment, starts it, waits until it is ready and prints its login link. It reuses the installed licence, terms agreement and hostname, as `bbx setup` does. On `localhost` it

uses local TLS; any other hostname uses the installed certificate pair in `~/sslcerts`. The certificate authority must be trusted on the machine running the customer's browser, not inside the container.

13.2 Docker engines and platforms

`bbx docker` reuses a working Docker engine and installs one only when none is available:

- **Linux:** installs Docker's official engine and the Compose plugin. AMD64 and ARM64 hosts run the image natively.
- **macOS:** uses a running Docker engine such as Docker Desktop; on Apple Silicon it runs the native ARM64 image. Without one, it installs the Docker CLI, Compose and a dedicated Colima virtual machine through Homebrew that matches the Mac: native ARM64 on Apple Silicon (profile `browserbox-arm64`), AMD64 on Intel (profile `browserbox`). A deployment created on an existing AMD64 VM keeps using it. Other Docker contexts are left unchanged.
- **Windows:** run the Unix `bbx` in WSL with Docker integration (Docker Desktop, or Docker inside WSL when systemd is available).

Each deployment remembers its engine. Set `BBX_DOCKER_CONTEXT` to a named Docker context before creating a deployment to choose the engine; later commands address that same engine and report it if it is unavailable, rather than recreating the deployment elsewhere. A transient `DOCKER_HOST` is refused for this reason.

Setup always checks the image with the registry, reuses layers that are already present, verifies the image and its platform, and saves its immutable digest. Later starts keep using that exact image until you update it explicitly.

13.3 Commands

Command	Effect
`bbx docker setup NAME [options]`	Configure without starting. Stop an existing deployment before changing it.
`bbx docker run NAME [options]`	Set up a new deployment and start it. An existing deployment accepts only `NAME` and starts its saved configuration.
`bbx docker start NAME`	Start the saved configuration, wait for readiness and print the login link.
`bbx docker stop NAME`	Stop gracefully, keeping all data.
`bbx docker restart NAME`	Stop, then start.
`bbx docker status NAME [--json]`	Show container and service health.
`bbx docker logs NAME [--follow]`	Show startup and runtime logs.
`bbx docker link NAME`	Check readiness, then print the login link.
`bbx docker update NAME --version X.Y.Z`	Pull and verify that release's image, then recreate the container with its data. `--image REFERENCE` names another image instead.
`bbx docker remove NAME`	Remove the container and network, keeping data.
`bbx docker remove NAME --purge`	Also delete the deployment's data and configuration.

`bbx docker config NAME`	Print the saved Compose project (it refers to secret files; it does not contain their values).
`bbx docker exec NAME bbx policy show`	Run a normal `bbx` command inside the deployment, as its service user.

`bbx docker help` prints the same summary. Login links are reusable credentials: print them deliberately and treat them like passwords. Container logs redact product keys and login tokens.

13.4 Setup options

Option	Meaning
`--hostname HOST`	Browser-facing hostname. Default: the installed hostname.
`--port PORT`	Main service port, as in `bbx setup`. Default: 8888.
`--bind IPV4`	Host address the ports are published on. Default: `127.0.0.1`.
`--tls MODE`	`local`, `provided` or `external` (below).
`--cert-dir DIR`	Directory holding `fullchain.pem` and `privkey.pem`, for `--tls provided`.
`--license-file FILE`	Product key file, copied privately into the deployment.
`--version VERSION`	Exact release image to use, saved by digest.
`--image REFERENCE`	Another image implementing the BrowserBox container interface.
`--accept-terms`	Accept `https://dosaygo.com/terms.txt` without a prompt.

`--license-file`, `--accept-terms`, `--tls` and `--cert-dir` are for headless or CLI-only hosts that have no BrowserBox installation to reuse. For example, a server with its own certificate:

```
bbx docker run work --hostname browser.example.com --port 8888 \
  --bind 0.0.0.0 --tls provided --cert-dir /etc/browserbox/certs \
  --license-file /etc/browserbox/license.key --accept-terms
```

Setup checks the certificate's hostname, expiry and matching key. Root-owned source files are supported. The licence and certificate pair are copied into the container for its service user; no certificate-authority private key is copied.

13.5 Ports and TLS modes

A deployment publishes the same four ports as a host installation, with equal host and container port numbers:

Service	Offset	Default port
Audio	−2	8886
Documents	−1	8887
BrowserBox	0	8888
DevTools (authenticated)	+1	8889

`--bind` controls where they are published (loopback by default); `--hostname` controls the identity the customer's browser uses. Chrome's private debugging port is never published.

- **`local`:** BrowserBox-managed local certificates, for `localhost`.

- **provided:** your certificate pair from `--cert-dir`.
- **external:** the services speak plain HTTP on host loopback, at the same four ports. Your reverse proxy must terminate HTTPS for all four public ports and forward the Host header and WebSocket upgrades. Rewriting only the main login URL is not enough.

WebRTC remains the default transport, with its usual WebSocket fallback. Published TCP ports do not guarantee direct UDP connectivity through every Docker or customer network; the fallback covers that case.

13.6 Data, licence and updates

Each deployment is an ordinary Compose project, saved as `compose.json` in a directory named after the deployment under `~/.config/dosaygo/bbx-docker/` (`BBX_DOCKER_HOME` changes that parent directory). It has one container and one named volume holding that deployment's browser profile, policy, configuration and downloads. The host's own BrowserBox configuration is never mounted.

- `stop` and `remove` keep the data; only `remove --purge` deletes it.
- Never attach one deployment's volume to two running containers, and do not copy volumes to create more seats. Every running deployment uses a licensed seat.
- The product key is stored as a private file (mode 0600) beside the project and mounted as a file; it does not appear in the container's configuration.
- `update` keeps the previous project as `previous-compose.json` for inspection. It does not roll back automatically if the new image fails to become ready; check data compatibility before returning to an older image.

Because the project is plain Compose, Docker's own tools also work on it, using the deployment's saved context:

```
docker compose -p bbx-work -f ~/.config/dosaygo/bbx-docker/work/compose.json up -d --wait
docker compose -p bbx-work -f ~/.config/dosaygo/bbx-docker/work/compose.json stop
```

The image never updates its own binary in place; change versions with `bbx docker update`.

13.7 Running the image on your own infrastructure

The image can be deployed directly with your own orchestration. It reads its configuration from the environment:

Variable	Meaning
`LICENSE_KEY_FILE` or `LICENSE_KEY`	Product key, from a mounted file or a platform secret. Never bake it into an image.
`BBX_TEST_AGREEMENT=true`	You accept the BrowserBox terms.
`PORT`	Main service port; or, with `BBX_SERVICE_HOSTS`, the single HTTP ingress port.
`APP_PORT`	Internal main port behind that ingress. Default: 8888.
`BBX_SERVICE_HOSTS`	JSON object with exactly the keys `main`, `audio`, `docs` and `devtools`, each a distinct HTTPS hostname.

`LOGIN_TOKEN`	Optional initial login token, 32–128 hexadecimal characters. A persisted deployment keeps its configured token.
`BBX_PUBLIC_URL`	Optional public HTTPS origin for a direct deployment.

Without `BBX_SERVICE_HOSTS`, the container serves the four adjacent ports directly. With it, the container serves all four services through one plain-HTTP ingress port behind your HTTPS edge, routing by hostname:

```
{
  "main": "browser.example.com",
  "audio": "audio.example.com",
  "docs": "docs.example.com",
  "devtools": "tools.example.com"
}
```

All four hostnames must reach the same instance on port 443. The ingress opens only after all four services are healthy; unknown hosts receive HTTP 421. If you set `LOGIN_TOKEN`, the login link is `https://<main-host>/login?token=<token>`, so nothing has to be read back from logs.

On Google Cloud Run, the service host map is required and the image switches to Cloud Run's HTTP transport automatically. Map the four hostnames to the one service (Google recommends an external Application Load Balancer for production), enable session affinity, allow CPU between requests, and run one instance per service: each instance is a separate browser, so instances must not be load-balanced behind one login link. Cloud Run's filesystem is ephemeral; persistent data needs a per-instance volume, never one shared between live instances. The container handles signals and graceful shutdown itself, releasing its seat on normal stop.

13.8 Hyperscale in the desktop app (beta)

Open **Hyperscale** (Ctrl/Command+6) in the desktop app (Section 4.3). Its **Local** view manages named Docker deployments on Linux and macOS. On Windows, use `bbx docker` from WSL. The **Cloud** view describes assisted Google Cloud Run deployment; it does not provision cloud resources from the desktop app. Contact `support@dosaygo.com` for cloud setup assistance.

1. Choose a saved deployment card, or choose **New deployment...**. Scroll the card list to reach additional deployments. Each name has separate browser data.
2. Use **Settings...** to review the name, hostname, port and other setup options, then **Save settings**. Saving configuration and starting the deployment are separate actions.
3. Choose **Start**, then **Check status** to obtain the deployment's observed state and login link. Use **Open** or **Copy login link** when enabled. **Logs** shows command output for diagnosis.
4. Use **Stop...**, **Restart...**, or **Remove...** only after reviewing the named deployment and consequence. Stop retains browser data. Remove retains saved settings, browser data and the licence; purge remains in All commands.
5. Use **Update image...** to choose an exact release or image. Browser data is retained. The full Docker command catalogue also provides advanced operations such as `exec`, `config`, and `remove --purge`.

The panel uses the same `bbx docker` commands as the terminal. Reading saved deployment cards does not establish that their containers are healthy; use Check status for that observation.

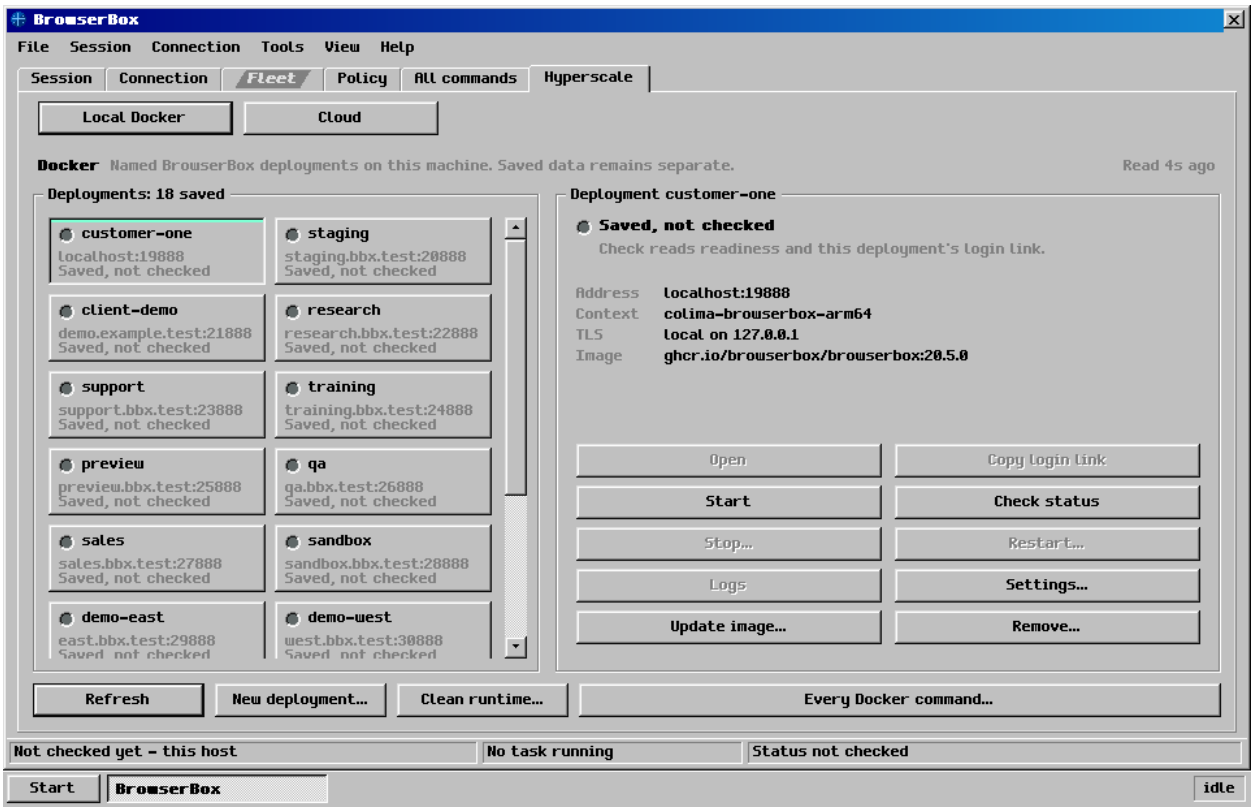


Figure 5: Hyperscale Local: saved deployment cards and actions for the selected deployment.

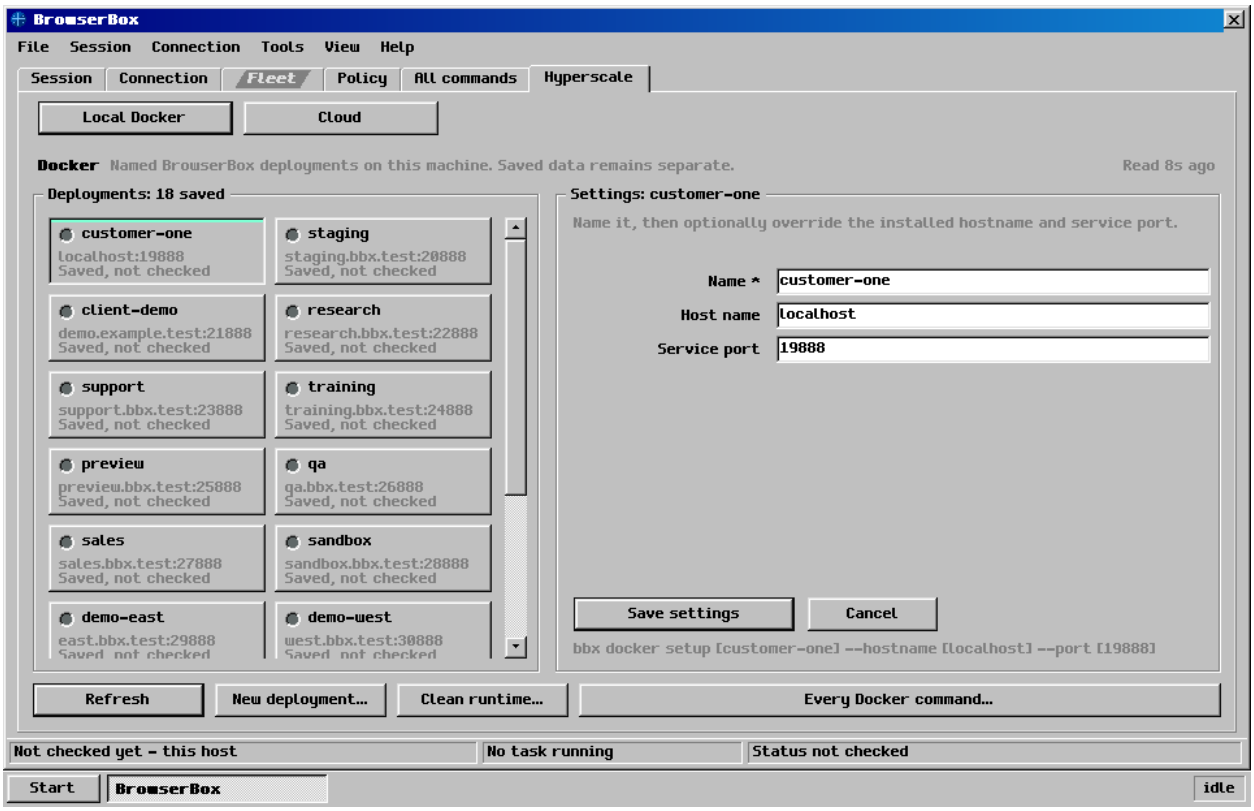


Figure 6: Deployment settings: review fields and the command preview before saving.

Docker engine operations are explicit because they may install an engine. Session's Stop controls the host BrowserBox; Hyperscale's Stop controls only the selected deployment.

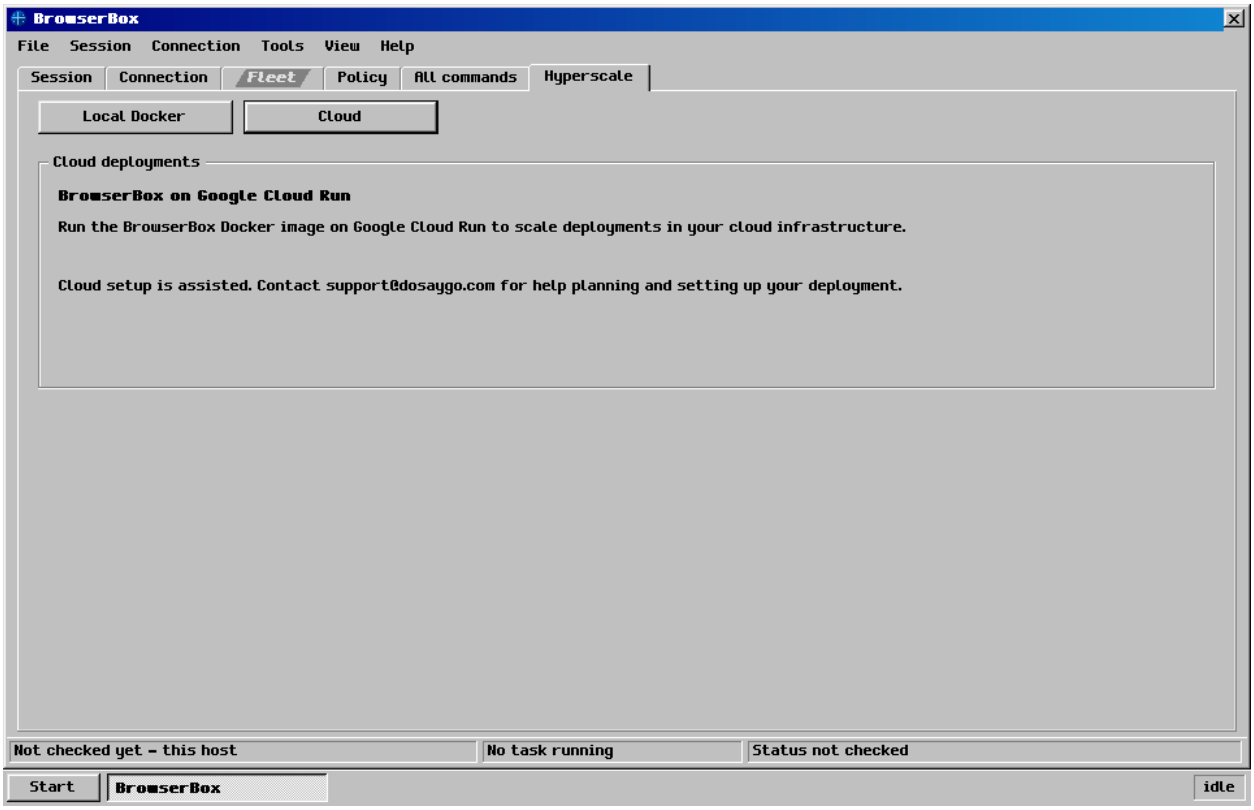


Figure 7: Hyperscale Cloud: assisted Google Cloud Run setup guidance.

13.9 Docker environment variables

`BBX_DOCKER_CONTEXT`	Named Docker context for a new deployment.
`BBX_DOCKER_IMAGE`	Image reference for a new deployment, instead of `ghcr.io/browserbox/browserbox:latest`.
`BBX_DOCKER_REPOSITORY`	Registry repository used by `--version`.
`BBX_DOCKER_HOME`	Parent directory for saved deployments.
`BBX_DOCKER_COMPOSE_VERSION`	Exact Compose plugin version installed on Linux.
`BBX_CLI_BIN_DIR`	Install directory for `install.sh --cli-only`.

14 TLS and Certificates

14.1 BrowserBox-managed TLS

On normal local or self-hosted setups, `bbx setup` ensures host resolution is sane, obtains certificates through BrowserBox's TLS flow, and starts BrowserBox in HTTPS mode when the expected certificate files exist.

BrowserBox automatically chooses the certificate strategy based on the hostname:

- **Domain names with public DNS** — Let's Encrypt via `certbot` (`http-01` challenge). An email address is required.
- **Public IP addresses** (e.g. `198.51.100.42`) — Let's Encrypt short-lived certificates (6-day validity) via `acme.sh`. Port 80 must be reachable. Falls back to `mkcert` if LE issuance fails.
- **Private/loopback IPs** (`127.x`, `10.x`, `192.168.x`, `172.16--31.x`) — locally trusted certificates via `mkcert`.
- **Local hostnames** (`localhost`, `*.local`, `*.test`, etc.) — locally trusted certificates via `mkcert`.

An existing certificate pair in `~/sslcerts/` (or `BBX_SSLCERTS_DIR`) is reused whenever it is unexpired, its key matches the certificate, and its subject alternative names cover the configured hostname; a wildcard covers exactly one label. When a new certificate is needed, it is issued into a private temporary directory and installed only after it is verified, so a failed issuance leaves the existing pair in place.

Example with a public IP:

```
bbx setup --hostname 198.51.100.42 --port 8888
# Obtains a Let's Encrypt short-lived cert (auto-renewed by acme.sh)
bbx start
```

14.2 Customer-managed certificates

If you want BrowserBox to use your own certificate material instead of BrowserBox-managed certificates, set:

```
export BBX_SSLCERTS_DIR="/path/to/your/certs"
```

The certificate directory must contain:

- `fullchain.pem`
- `privkey.pem`

When `BBX_SSLCERTS_DIR` is set and the directory already contains `fullchain.pem` and `privkey.pem`, BrowserBox skips all certificate generation (no Let's Encrypt, no mkcert) and uses the provided certificates as-is.

When `BBX_SSLCERTS_DIR` is set but the directory does not yet contain certificates:

- `bbx setup` and `bbx ng-run` generate certificates into that directory instead of `~/sslcerts/`. `bbx ng-config validate` checks the existing pair but never generates one.
- Nginx configuration points to the override directory.
- The BrowserBox runtime reads certificates from there.
- In `--for` mode (Section 10), certificates are generated into the override directory and `chown`ed to the target user.

If you already have externally managed certificates, simply point `BBX_SSLCERTS_DIR` at the directory—BrowserBox detects existing certs and skips generation automatically.

For `bbx ng-run`, BrowserBox generates per-service names as `<random>.<domain>`. The certificate must cover those generated names. For example:

```
BBX_HOSTNAME=bbx-dev.corp.example
required nginx SAN: *.bbx-dev.corp.example
```

A shallower wildcard such as `*.corp.example` only covers one label under `corp.example`. When `BBX_SSLCERTS_DIR` points at an existing certificate and exactly one shallower wildcard parent matches the configured hostname, BrowserBox uses that parent for `ng-run` route generation and logs the decision. You can also make the parent explicit:

```
export DOMAIN="corp.example"
bbx ng-run --for bbxruntime
```

Another common TLS pattern is to run `bbx setup --backend http` and terminate TLS in nginx, Caddy, or another reverse proxy.

15 Customer-Relevant Environment Variables

15.1 Core customer-facing variables

The native BrowserBox runtime accepts the established `user.env` and `test.env` contract. Unknown variables are ignored, and punctuation in string-valued settings is treated as data rather than evaluated as program text. The table lists active customer controls. Obsolete tuning aliases that are accepted only to keep an older environment file from blocking an upgrade are intentionally omitted.

Name	Default	Meaning
`ALLOWED_EMBEDDING_ORIGINS`	unset	Space-separated allowlist of parent origins permitted to embed BrowserBox.
BBX_HTTP_ONLY	unset	Forces HTTP-only mode, skips direct DNS ownership checks, and skips HTTPS certificate generation. Useful behind a reverse proxy.
BBX_EXTERNAL_TLS	unset	Treats the frontend as secure when TLS is terminated upstream. When exported before installation, also skips installer-time BrowserBox certificate generation.
BBX_NO_HSTS	unset	Explicitly set to `true` (also accepts `1`, `yes`, or `on`) to omit only the HTTP Strict Transport Security header. This is independent of Tor and TLS mode; use it for a corporate ingress that cannot accept HSTS. HTTPS, certificate validation, CSP, and every other security header remain enabled.

BBX_SSLCERTS_DIR	unset	Directory containing customer-managed TLS files such as `fullchain.pem` and `privkey.pem`. Respected by `setup`, `ng-run`, `ng-config`, and `--for` mode.
BBX_SHUTDOWN_IDLE_MS	2700000	Main no-client shutdown from runtime startup and after the last client disconnects.
`BBX_CONNECTED_IDLE_MS`	0 (disabled)	Native runtime shutdown after this period without accepted activity while a client remains connected. Accepted input, advancing active-tab video and measured non-silent audio renew the deadline. Maximum 7 days.
`BBX_CONNECTED_IDLE_WARNING_MS`	half the idle period, capped at 60000	Notice lead time before connected-idle shutdown. Set to 0 to disable the warning; must be less than the idle period.
`BBX_CONNECTED_IDLE_WARNING_MESSAGE`	built-in notice	Custom UTF-8 notice body, up to 4096 bytes.
`BBX_MINIMAL_MODE_IDLE_MS`	120000	Minimal-mode idle shutdown fallback.
`FLEET_REAP_INTERVAL_SECS`	5	Delay between foreground `fleet monitor` recovery passes.
`FLEET_REAP_GRACE_SECS`	15	Required continuously-down window before monitor or one-shot reaping releases an allocation.
`FLEET_ACQUIRE_REAP_GRACE_SECS`	2	Confirmation window for acquire-time recovery of allocations already observed fully down.
`FLEET_STALE_RESERVE_SECS`	900	Age after which a fully down transitional allocation is eligible for monitor/reap recovery.
`FLEET_MONITOR_MAX_BACKOFF_SECS`	300	Maximum retry delay after repeated monitor-pass failures.
`FLEET_MONITOR_FAILURE_DETAIL_LIMIT`	10	Maximum per-allocation automatic-release failure lines in one pass; additional failures are summarized.
BBX_HOME_PAGE	unset	Preferred home page URL.
`BBX_DEFAULT_HOME_PAGE`	https://duckduckgo.com	Fallback home page URL.

BBX_CUSTOM_CURSOR	unset	Path to a local image file used as the remote cursor image.
BBX_DOC_CONVERSION_MAX_SECONDS	120 seconds	Maximum time allowed for each document probe or secure-view conversion. Set a whole number from 1 to 86400; for example, 600 allows ten minutes for a large PDF. Overrides the older BBX_DOCS_JOB_TIMEOUT_MS setting when present. Invalid values prevent document service startup. Persist in user.env and restart BrowserBox.
BBX_HOSTNAME	system hostname or prompt	Hostname used for BrowserBox setup.
DOMAIN	unset	Explicit route-domain override for network-oriented commands such as ng-run. Use only when overriding the saved setup domain.
BBX_UI_THEME_OVERRIDE	unset	Default login UI theme when a supported theme is requested.
BBX_SHOW_STREAMING_STATS_OVERLAY	unset	Displays the live streaming stats overlay in the BrowserBox client UI.
BBX_MAX_CONNECTIONS	3	Per-session client connection cap, from 1 through 5. Clients admitted with the same connector co-browse the same remote browser. Larger positive legacy values are safely capped at 5.
BBX_MAX_TABS	20	Maximum concurrent page tabs per BrowserBox session. Applies to both user-initiated tab creation and browser-initiated popups (e.g. window.open or links with target="_blank"). When a client is connected at the time a popup is blocked, a notification appears in the client UI. Tabs that would exceed the cap at service startup are closed silently, with no client notification. Set this persistently in user.env (see Section 8.4) so it survives bbx setup regeneration.
BBX_CLEAN_SLATE	unset	When true, clears the browser user-data directory at startup. Useful for ephemeral deployments that need a fresh profile every launch.

BBX_MINIMAL_MODE	unset	Runs BrowserBox with only the main service (no audio, docs, or devtools sidecars). Useful for constrained environments or single-purpose kiosks.
BBX_DISABLE_WEBRTC	unset	When `true`, disables WebRTC entirely and forces WebSocket-only transport.
`BBX_DISABLE_WEBRTC_RACING`	false	Native BrowserBox starts with WebRTC and retains WebSocket fallback, but normally sends one sparse same-frame sample every 60 published frames so the client can compare both paths. Set to `true` to disable only those samples and learned path switching; it does not disable WebRTC.
BBX_AD_BLOCK	true	Master switch for ad-specific blocking. Set to `false` and restart BrowserBox to disable both BrowserBox request-level ad blocking and Chrome's native ad filter. Policy enforcement, authentication, and PDF handling may still use request interception without blocking ads.
`BBX_NATIVE_AD_BLOCK`	false	Adds Chrome's native subresource ad filter when `true`. Has no effect when `BBX_AD_BLOCK=false`.
BBX_LOW_END_MODE	true	When `true`, forces Chrome low-end-device behavior. Set to `false` on adequately provisioned performance hosts; validate constrained deployments before changing it.
BBX_NO_AUDIO	unset	When `true`, does not start the audio service and tells the client not to retry audio setup. Use only when remote audio is not required.

BBX_NETWORK_DOMAIN	false	Controls fine-grained CDP Network tracking. Disabled by default: BrowserBox synthesizes page-loading progress from page lifecycle events, avoiding a high-volume internal event stream on network-heavy applications. Set to `true` to restore fine-grained network telemetry. Navigation to local files and browser-internal pages remains blocked in both modes.
BBX_OOPIF	true	Attaches BrowserBox to Chrome's separately rendered cross-site iframe targets so cursor and supported page integrations work inside OOPIFs. Set to `false` and restart only as a temporary compatibility fallback.
BBX_GPU	unset	On supported bare-metal Linux hosts, `true` enables the hardware-GPU profile, including explicit headless GPU enablement, native EGL ANGLE, and GPU rasterization.
`BBX_SUPPORT_CONTACT`	support@dosaygo.com	Email address or URL shown as the support contact on built-in blocked pages (Section 5.8).
`BBX_STATUS_PAGES_DIR`	unset	Additional directory of customer blocked-page files, searched first (Section 5.8).
BBX_NO_UPDATE	unset	Disables automatic and explicit updates (Section 3.3).
BBX_NO_CDN	unset	When `1`, downloads release files from GitHub only, bypassing the release mirror.
BBX_ASSET_BASE	https://dl.getbrowserbox.com	Base URL of the release mirror. Point it at your own mirror laid out as `<base>/<tag>/<file>`.
`BBX_PROGRESS_EVENTS`	unset	When `1`, long-running commands print `@bbx-progress` and `@bbx-failure` markers (Section 4.4).
`BBX_DEBUG_LICENSE`	unset	When `true`, logs routine licensing requests for support diagnostics.
LICENSE_KEY	unset	Your BrowserBox product key. Can be set in the environment, passed to `bbx certify`, or persisted in `test.env` after first certification.

`HTTP_PROXY / HTTPS_PROXY` `http_proxy / https_proxy`	unset	Standard forward-proxy URLs. BrowserBox runtime HTTPS calls, including continuous licence validation, honour these values.
`NO_PROXY` `no_proxy`	unset	Standard comma-separated proxy exclusions. Include loopback and internal domains/ranges even though BrowserBox's own loopback health checks bypass proxies explicitly.
FLXTLS_CA_BUNDLE	unset	PEM bundle containing additional trusted CAs, commonly the corporate TLS-inspection CA. Set before `bbx start`; never disable TLS verification to accommodate interception.

HSTS opt-out scope. BrowserBox emits HSTS by default. `BBX_NO_HSTS` does not enable HTTP, weaken certificate checks, or change Tor behavior; it only omits `Strict-Transport-Security` from BrowserBox responses. A browser that previously cached HSTS for the hostname may retain that state until its cached policy expires or is cleared, so set this before first production access on an affected corporate hostname.

15.2 Examples

```
export ALLOWED_EMBEDDING_ORIGINS="https://app.example.com https://localhost:*"
export BBX_HTTP_ONLY=1
export BBX_EXTERNAL_TLS=true
export BBX_SSLCERTS_DIR="/path/to/certs"
export BBX_HOME_PAGE="https://intranet.example.com"
export BBX_CUSTOM_CURSOR="/absolute/path/to/cursor.png"
export BBX_SHUTDOWN_IDLE_MS=7200000
export BBX_CONNECTED_IDLE_MS=1800000
export BBX_CONNECTED_IDLE_WARNING_MS=60000
export BBX_UI_THEME_OVERRIDE=dark
export BBX_SHOW_STREAMING_STATS_OVERLAY=true
export BBX_MAX_CONNECTIONS=5
export BBX_MAX_TABS=24
bbx setup
bbx start
```

Cross-user execution:

```
# Run as operator, services execute as bbxruntime
bbx setup --port 9090 --hostname app.example.com -z --for bbxruntime
bbx ng-run --for bbxruntime
bbx stop --for bbxruntime
```

Certificate override with cross-user execution:

```
export BBX_SSLCERTS_DIR="/opt/shared-certs"
bbx ng-run --for bbxruntime
```

16 Licensing and Occupancy Visibility

16.1 Available today

- `bbx certify` validates the license and reserves a seat when possible.
- `bbx vacancy` queries the license server and reports total seats plus current occupied and vacant counts when the server supports occupancy summaries.
- Local reservation metadata is stored under `~/.config/dosaygo/bbpro/tickets/`.
- A running BrowserBox revalidates its licence every 331 seconds, using the fallback licensing endpoints when a primary is unavailable (Section 3.6).

16.2 When every seat is in use

If certification or startup finds no vacant seat, `bbx` stops with:

```
Error: No vacant seat available. Every seat on this licence is in use.

  See who holds them:  bbx vacancy
  Free one:            bbx stop-user <username>
  Add seats:           sales@dosaygo.com

Extra seats are added to your existing licence; your key does not change.
```

Additional seats are added to the existing licence by DOSAYGO sales; the product key stays the same and no reconfiguration is needed. `bbx stop-user` is not available on Windows; stop the BrowserBox instance holding the seat on that machine with `bbx stop` instead.

16.3 Ticket upgrade on first certification

Seat reservations (tickets) now carry a signing key generated on the BrowserBox host. The first certification after upgrading from an older release converts an existing ticket automatically and prints `Updating existing ticket to holder-owned signing credentials...` No action is required. If certification instead reports `Existing ticket cannot be migrated`, contact support@dosaygo.com with the `bbx certify` output.

16.4 What `bbx vacancy` means

```
bbx vacancy
```

For modern servers, `bbx vacancy` reports a customer-safe summary:

- total seats
- occupied seats now
- vacant seats now
- leased active
- reserved active

By default it does not expose seat IDs or deeper lease inventory to ordinary customer keys.

16.5 What is not surfaced here

This BrowserBox runtime still does not surface richer customer history such as:

- detailed current lease inventory for ordinary customer keys
- free-seat counts over time
- occupancy history

17 Practical Recipes

17.1 Embedded BrowserBox without the standard UI

```
export ALLOWED_EMBEDDING_ORIGINS="https://app.example.com"
export BBX_HOME_PAGE="https://app.example.com"
bbx setup -p 8888
bbx start
```

Then use a login link shaped like:

```
https://localhost:8888/login?token=...&ui=false
```

17.2 BrowserBox behind a reverse proxy

```
bbx stop
bbx setup --backend http -p 8080 --hostname browserbox.internal
bbx start
```

Then let nginx, Caddy, or a cloud load balancer terminate TLS and forward plain HTTP to BrowserBox.

If the deployment needs audio or DevTools, the edge must preserve a distinct origin per service rather than folding them onto one hostname — see Section 8.6 for the required layout.

17.3 Check whether a seat is currently available

```
bbx vacancy
```

This command reports the current seat summary, including total seats, occupied seats, vacant seats, active leases, and active reservations.

Some deeper occupancy reporting may vary by deployment version.

18 Transport and Streaming

BrowserBox uses an ACK-paced, bounded streaming pipeline to deliver remote browser frames to connected clients. Two transport modes are available:

18.1 WebRTC (default)

By default, BrowserBox negotiates a peer-to-peer data channel between server and client. WebRTC is the initial path. One of every 60 published frames is eligible for a same-frame WebRTC/WebSocket comparison; the client reports the faster path, and native BrowserBox requires two consistent observations before switching so a single jittery sample cannot flap the selection.

18.2 WebSocket fallback

If WebRTC negotiation fails (firewalled UDP, restricted NAT, or `BBX_DISABLE_WEBRTC=true`), BrowserBox falls back to WebSocket binary transport. This works through HTTP proxies and load balancers that support WebSocket upgrades and still delivers real-time frames. BrowserBox does not retry WebRTC indefinitely after a bounded failure.

For deployments behind an HTTP proxy or strict corporate firewall where WebRTC is unreliable, forcing WebSocket-only mode improves connection stability:

```
export BBX_DISABLE_WEBRTC=true
bbx start
```

To retain WebRTC and WebSocket fallback but disable the sparse comparison and learned switching:

```
export BBX_DISABLE_WEBRTC_RACING=true
bbx start
```

18.3 Capture profile and backpressure

Native v20 intentionally uses the release-readiness profile measured during cutover: JPEG quality 80 and every captured frame. Backpressure remains adaptive even though JPEG quality is fixed:

- Each viewer has bounded frame and transport ownership; capacity refusal follows a typed fallback path rather than terminating the service.
- When frames back up, newer state can coalesce over stale unsent state so recovery favors the current page rather than replaying an old queue.
- Round-trip acknowledgements, credits, mailbox/coalescing counts, selected transport, dimensions, and frame byte length feed the existing streaming diagnostics overlay.

18.4 Streaming diagnostics

BrowserBox can show a live streaming stats overlay in the client UI with FPS, transport mode, round-trip time, and frame-flow status.

To display it:

```
export BBX_SHOW_STREAMING_STATS_OVERLAY=true
bbx start
```

The overlay shows live metrics including:

Metric	Meaning
FPS	Frames per second currently being delivered to the client. This reflects actual browser redraw activity—if the remote page is static, the FPS will naturally be low even on a healthy connection.
Transport	Active transport mode: `WebRTC` or `WebSocket`.
RTT	Round-trip time between server and client, used by the pipeline for pacing decisions.

Backpressure	Whether the pipeline is currently throttling or dropping frames to keep up with network conditions.

Important: a low FPS reading does not necessarily indicate a problem. If the remote page content is not changing, BrowserBox does not generate frames—FPS measures browser redraw activity, not a minimum guaranteed stream rate. A high-resolution page with no animations will show near-zero FPS on a healthy connection.

18.5 Remote interaction

- **Double and triple clicks.** BrowserBox counts rapid presses the way the local platform does and sends the count to the remote page, so double-click handlers, double-click word selection, and triple-click line selection work. Mouse presses count as a sequence within 500 ms and 4 pixels; touch and pen taps within 400 ms and 24 CSS pixels.
- **Dropdowns.** A single-choice `<select>` on the remote page opens the user's own operating-system dropdown, positioned over the remote control, so it looks and behaves natively on desktop and mobile. Multi-select lists and list boxes (`size` greater than 1) are operated directly on the remote page.

19 Flipbook Recording

BrowserBox can record a browsing session as a *flipbook*—a self-contained static site of sequential JPEG frames with a JavaScript viewer. Flipbook recordings are produced directly from the internal screencast pipeline with negligible overhead: when recording is off, the only cost is a single boolean check per frame.

19.1 Quick start

```
bbx setup --flipbook-record ~/my-recording \
          --flipbook-description "Onboarding walkthrough"
bbx run
# ... use BrowserBox normally ...
bbx stop
```

When `bbx stop` is called, BrowserBox compiles the captured frames into a complete flipbook static site and (if Cloudflare `wrangler` is available) offers to deploy it to Cloudflare Pages.

19.2 How it works

1. `bbx setup` writes `BBX_FLIPBOOK_DIR` and optionally `BBX_FLIPBOOK_DESCRIPTION` to `~/.config/dosaygo/bbpro/test.env`.
2. At runtime, each screencast frame is written as a JPEG + JSON metadata pair to a temporary directory inside the BrowserBox config directory (`~/.config/dosaygo/bbpro/flipbook_recording/`).
3. On `bbx stop`, the built-in `browserbox flipbook-generate` command compiles the raw frames into a flipbook static site under the directory provided by `--flipbook-record`.

When recording is disabled (no `--flipbook-record` flag), **there is zero overhead** on the frame pipeline—only a single falsy-value check is evaluated per frame.

19.3 Output structure

Each recording produces a timestamped directory inside the flipbook base directory:

```
~/my-recording/
  2026-04-14T02-25-00-000Z--2026-04-14T02-30-00-000Z/
    site/
      index.html          # self-contained viewer
      manifest.json       # flipbook v1 manifest
      pages/
        000000.jpg        # contiguous, zero-padded frames
        000001.jpg
        ...
      assets/
        viewer.css
        viewer.js
        sw.js             # service worker for offline caching
      meta/
        provenance.json   # full per-frame metadata + recording info
```

Multiple runs with the same `--flipbook-record` directory produce separate timestamped subdirectories, each a self-contained flipbook site.

19.4 Frame normalization

Frame IDs from the internal pipeline may contain gaps (due to cast restarts or tab switches). The site generator performs a normalization pass: frames are sorted, paired by basename (JPEG with its JSON metadata), and re-indexed as a contiguous zero-padded sequence (`000000`, `000001`, ...). The original frame IDs are preserved in `meta/provenance.json` for traceability.

19.5 Cloudflare Pages deployment

If `wrangler` (the Cloudflare CLI) is available or can be installed, `bbx stop` will offer to deploy the generated flipbook site to Cloudflare Pages:

```
# Install wrangler if not already present
npm install -g wrangler

# Authenticate (one-time)
wrangler login

# Deployment happens automatically on bbx stop
```

The Pages project name is derived from the flipbook directory basename. If `wrangler` is not available, the site is still generated locally and can be served with any static file server.

19.6 Relevant environment variables

`BBX_FLIPBOOK_DIR`	Absolute path to the flipbook output directory. Set by `--flipbook-record`. When empty, recording is disabled.

`BBX_FLIPBOOK_DESCRIPTION`	Optional human-readable description embedded in the manifest and provenance metadata. Set by `--flipbook-description`.

19.7 Direct site generation

The flipbook site generator is available as a standalone command on the BrowserBox binary:

```
browserbox flipbook-generate <flipbookDir>
```

This reads raw frames from the config directory, compiles the flipbook site into a timestamped subdirectory of `<flipbookDir>`, and cleans up the temporary recording data. This is called automatically by `bbx stop` but can also be invoked manually.