# KLIPKAST

# Broadcaster MAX

## User & Administrator Whitepaper

*Version 3 · Text-only edition · Golden Baseline v64.0.78*

Self-hosted HLS streaming · DVR · PlayBox playout · Multi-protocol output

# Contents

## Part I

# User Guide

*Everything a broadcaster or producer needs to run day-to-day operations*

# 1. Introduction

KlipKast Broadcaster MAX is a self-hosted streaming server that turns a single Linux box into a full broadcast network. This guide walks you through the day-to-day workflow: signing in, creating streams, scheduling playout with PlayBox, pushing to social platforms, and reviewing archives.

# 2. Signing In

Open a browser to the KlipKast address supplied by your administrator, typically:

```
https://ip_address/
```

Sign in with the operator account issued to you. The header shows your username and the server you are connected to.

# 3. Dashboard Overview

The Dashboard is the single-glance view of your broadcast network. Summary cards show live streams, viewers, outbound bandwidth, and uptime. The "Currently Live" table lists every source that is actively on-air, and the right rail shows real-time system health for CPU, RAM, ramdisk, and the service ports.

## 3.1 Reading the status pills

- LIVE — source is publishing and viewers can connect
- STANDBY — a template or PlayBox loop is filling the slot until a real source returns
- OFFLINE — source is intentionally stopped
- ERROR — source failed health-check; see Health page for detail

# 4. Streams

The Streams page lists every ingest you have configured. From here you create new streams, search by name, filter by type or status, and open any stream to see its full detail view.

## 4.1 Creating a new stream

1. Click the "+ New Stream" button in the toolbar.
2. Give the stream a URL-safe name (letters, numbers, dashes, slashes). This becomes the last segment of every output URL.
3. Pick a source type: HLS Pull, RTMP Push-In, SRT, MPEG-TS, or Icecast.
4. Paste the source URL. For push-in types, KlipKast will show you the ingest URL and key to hand to the encoder.

5. Optionally attach a Template — this pre-fills output, destinations and overlays in one click.
6. Save. The stream starts immediately if "Auto-start" is enabled.

### 4.2 Supported source types

| Type | Direction | Typical use |
|---|---|---|
| HLS Pull | Server pulls | Ingesting an existing HLS feed from another CDN |
| RTMP Push-In | Encoder pushes | OBS, vMix, hardware encoders |
| SRT | Bi-directional | Low-latency contribution across the internet |
| MPEG-TS / UDP | Server pulls | On-prem broadcast plant feeds |
| Icecast | Server pulls | Audio-only radio streams |

## 5. Stream Detail

Opening any stream drops you into its detail view. The preview panel shows live output and media metadata (resolution, codec, FPS, audio, GOP, bitrate). Other panels list every publish endpoint and every push destination.

### 5.1 Endpoints

Every live stream is published on multiple protocols simultaneously. Share the URL that matches your viewer's player:

- HLS — browsers, iOS, Roku, most Smart TVs
- RTSP — IP-camera clients and low-latency players
- RTMP — legacy players and re-ingest

When token protection is enabled, viewer URLs are automatically routed through the token proxy on port 3003 and rotated by the server. Viewers never see the underlying token.

## 6. PlayBox — Scheduled Playout

PlayBox turns a folder of MP4 clips into a live 24/7 channel. Drop files into the playlist, bind the playlist to a stream slot, and PlayBox handles the rest: continuous looping, in-clip overlay burn-in, cinematic push-back transitions, and atomic file swaps so the output never blinks.

### 6.1 Building a playlist

7. Upload MP4/MOV clips into the Playlists page.
8. Drag them into the desired order. Duration is auto-detected.

9. Optionally attach an overlay to each clip — the overlay is burnt in during the render, not composited live, which is why it looks broadcast-clean.
10. Bind the playlist to a stream slot (for example the "willberightback/live" standby channel).
11. Choose Continuous loop, Scheduled start, or One-shot.

### 6.2 Overlays

Overlays live in a small library. Each overlay is a PNG or animated MOV with an alpha channel. Common examples: news lower-thirds, interview bugs, weather crawls, and emergency alerts. Overlays can be scheduled, recurring, or triggered live from the Overlays page.

## 7. Templates

Templates are reusable "recipes" that pre-configure a stream: output protocols, push destinations, overlays, DVR retention, and token refresh policy. Templates dramatically speed up onboarding: instead of configuring six panels, you pick a template and the stream is production-ready.

### 7.1 When to make a template

- You are running more than one stream with the same push destinations
- You use the same overlay style across multiple events
- You rely on a token-refresh source pattern (partner CDNs, licensed feeds)
- You need a repeatable "standby" configuration for holiday coverage

Golden Baseline v64.0.78 guarantees that template-created streams and their destinations survive reinstall and reboot. Do not disable the shadow-file safety net.

## 8. Push & Restream

Every stream can fan out to multiple RTMP/RTMPS destinations at once — YouTube, Facebook, Twitch, custom CDNs — without re-encoding. A supervisor process (push-fanout) reconciles the desired list every five seconds, reconnects on drop, and persists the destination list to disk.

### 8.1 Adding a destination

12. Open the stream detail and select the Push tab.
13. Click "Add destination".
14. Enter a label (freeform), the RTMP/RTMPS URL, and the stream key.
15. Click Test to verify handshake, then Save & Push.
16. Toggle Reconnect on. The supervisor will now maintain the push automatically.

### 8.2 Destination persistence

Destinations are stored in streams.json and mirrored to a shadow file (push-destinations-backup.json). Even if a stream is recreated by template refresh, its destinations are rehydrated from the shadow. This behavior is a frozen guarantee of the v64.0.78 baseline.

## 9. Archives, DVR & VOD

Every live stream can automatically record to disk. Choose retention per stream: rolling window (e.g. 6 hours), scheduled recording, or continuous archive. Recordings appear on the Archives page as scrubbable segments.

### 9.1 Turning a recording into VOD

17. Open Archives and scrub to the in-point.
18. Set the out-point.
19. Click "Save as VOD" — the clip is remuxed to MP4 without re-encoding.
20. The clip appears in VOD and gets its own playback URL.

## 10. Live Now, Analytics & Viewers

Live Now is a wall-view of every on-air stream, useful in control rooms. Analytics shows historical viewer counts, bitrate stability, and drop-out heatmaps. The Viewers page lists active sessions and lets you kick abusive clients.

## 11. What to Do When Something Goes Wrong

The Health page is your first stop. It shows service status for all four supervisors, ramdisk usage, ghost-RAM guard state, and per-stream health scores. Most operator problems fit into three buckets:

### 11.1 A stream shows OFFLINE that should be on

- Check the source URL is reachable from the server (Health → Source Probe)
- Check push credentials have not expired
- For token-refresh streams, verify the refresh policy is still valid

### 11.2 Viewers report buffering

- Look at HLS Delivery (port 3002) throughput — is bandwidth saturated?
- Check ramdisk occupancy; tmpfs full will cause segment stutter
- Confirm the source itself is stable — Analytics shows bitrate variance

## 11.3 Push destination keeps reconnecting

- Verify the destination platform accepts the stream key
- Check whether the platform enforces per-key limits (YouTube, Twitch)
- Look at push-fanout logs on the Health page for the exact upstream error

## Part II

# Administrator Guide

*Installation, architecture, security, and operations for the person running the box*

# 12. Architecture Overview

KlipKast is a single-node broadcast pipeline. All state is JSON on disk under /opt/streamhub/data/ — there is no external database. Media flows through a two-stage pipeline (ingest → render → publish), buffered on a ram-backed tmpfs, and served by a hardened Node.js API alongside an FFmpeg-based delivery layer.

## 12.1 Port map

| Port | Role | Exposed to |
|---|---|---|
| 3001 | API and Control plane | Internal / admin UI |
| 3002 | HLS Delivery | Viewers (via nginx) |
| 3003 | HLS Token Proxy | Loopback only |
| 1935 | RTMP ingest / push | Encoders and downstream |
| 8554 | RTSP output | Low-latency clients |
| 443/80 | nginx TLS front door | Public |

Ports 3000 and 9090 are strictly forbidden — they clash with common third-party tooling.

## 12.2 The two-stage pipeline

- Stage 1 — Ingest and normalize: pull the source, decode when required, apply overlay burn-in, write MPEG-TS segments to tmpfs.
- Stage 2 — Publish: FFmpeg reads from tmpfs and emits HLS, RTSP, and RTMP simultaneously with -c copy where possible.
- Push fan-out is a third supervisor process that pulls the local HLS and pushes to each configured RTMP destination.

# 13. System Requirements

| Component | Minimum | Recommended |
|---|---|---|
| OS | Ubuntu 22.04 / 24.04 LTS / Debian 12 | Ubuntu 24.04 LTS |
| CPU | 4 cores | 8+ cores |
| RAM | 8 GB | 16 GB or more |
| Disk | 100 GB SSD | 500 GB NVMe |
| Network | 100 Mbps up | 1 Gbps up |

**Minimum system requirement**

Requirements: Ubuntu LTS (22.04 or 24.04) or Debian stable (12). 64-bit x86.
Non-LTS Ubuntu releases are not supported.

# 14. Installation

Installation is a single script executed as root. It pins the version, installs dependencies, lays down all services, and starts the supervisors.

```
curl -fsSL https://ip_address/install/streamhub-installer-v6478.sh | sudo bash
```

After the installer finishes:

21. Browse to https://ip_address/ and complete first-run setup.
22. Create the initial admin account.
23. Paste your license key on the Settings → License page.
24. Reboot to confirm services come up cleanly.

# 15. On-Disk Layout

```
/opt/streamhub/  backend/  server.js  ← authoritative backend  push-fanout.js  ← RTMP push supervisor  hls-token-proxy-service.js  ← port 3003  data/  streams.json  ← single source of truth  push-destinations-backup.json ← shadow safety net  templates/  overlays/  tmpfs/  ← ram-backed HLS segments
```

All backend files live under /opt/streamhub/backend/. Never edit server.js under any other path — patches applied elsewhere are ignored.

# 16. Services (systemd)

```
systemctl status streamhub  # main API + control systemctl status streamhub-push-fanout  # RTMP push supervisor systemctl status streamhub-hls-token-proxy  # port 3003 systemctl status nginx  # TLS front door
```

systemd is mandatory. Do not run the backend under pm2, forever, or a bare shell — the supervisor policies and ghost-RAM guard depend on unit-level restart semantics.

# 17. Persistence Model

There is no database. Every persistent value lives in JSON:

- streams.json — every stream record, including template_id, push_destinations, restream_destinations
- push-destinations-backup.json — shadow of destinations only, used to rehydrate recreated streams
- templates/*.json — one file per template

The shadow file is written whenever destinations change. On stream recreation (template refresh, RTMP resync) the restore path only fills keys that are entirely absent. It never overwrites live values. This behavior is frozen by v64.0.78.

# 18. Ghost-RAM Protection

Dead FFmpeg processes and abandoned tmpfs segments used to leak memory. v63 introduced a three-layer defense that is still active in v64:

25. Reaper — scans /proc for orphaned ffmpeg every 30 seconds and terminates the owning cgroup.
26. Guard — enforces a per-stream RAM ceiling. Exceeding the ceiling triggers a graceful restart.
27. Spawn-lock — prevents duplicate ffmpeg instances for the same stream ID.

If a stream ever reports "STANDBY" for longer than a minute without recovering, the guard has almost certainly caught a leak — check journalctl.

# 19. Token Refresh & Proxy

Partner feeds often use short-lived tokens embedded in HLS URLs. KlipKast performs zero-restart token swaps via a loopback proxy on port 3003:

- Tokens refresh at 90% of their advertised lifetime, with a 3-second minimum floor.
- New tokens are hot-swapped without dropping viewers.
- Corrupt refresh intervals on disk are auto-repaired at load time.

# 20. Backups

The entire state is a small JSON tree. Back it up on a schedule:

```
tar czf /var/backups/streamhub-$(date +%F).tgz \   /opt/streamhub/data
/opt/streamhub/backend/*.json
```

To restore, stop services, extract into place, and start again. Because state is JSON, restores are inspectable and diffable.

# 21. Upgrades & Version Policy

KlipKast versions use MAJOR.MINOR.PATCH where PATCH ranges 0–99. When PATCH hits .99 the MAJOR rolls over. Certain versions are frozen "Golden Baselines" and MUST NOT be patched in place — any fix bumps to the next version.

## 21.1 Currently frozen baselines

- v64.0.78 — full destination persistence (current)
- v64.0.53, v64.0.52, v64.0.49, v64.0.42
- v63.0.88, v63.0.85, v63.0.83, v63.0.74
- v62.0.98

If you find a bug in a frozen baseline, do not edit the file. Bump the version, apply the fix in the new file, and update VERSION_HISTORY.

# 22. Security

- Serve the UI and API exclusively behind nginx with TLS. HTTP-only exposure is not supported.
- Restrict port 3001 to loopback or the admin VLAN.
- Never expose port 3003 externally — it is a loopback token proxy.
- Roles are role-based: operator, admin, viewer. Store role assignments only in the user_roles table (or the JSON equivalent) — never on the user record.
- License key checks happen at every startup and on every stream toggle.

# 23. Monitoring

KlipKast surfaces metrics on the Health page and on the /api/health JSON endpoint. Recommended external monitors:

- HTTP probe on https://ip_address/api/health (expect 200 + JSON status:"ok")
- TCP probe on 1935 (RTMP ingest)
- Disk-free alarm at 1 GB (500 MB is the hard floor before ingest is paused)
- RAM alarm at 85% (primary bottleneck)

# 24. Troubleshooting Playbook

## 24.1 The UI loads but no streams start

- Check systemctl status streamhub — the API may be up while ffmpeg spawn is blocked
- Check disk-free is above 500 MB
- Check tmpfs is mounted (mount | grep streamhub)

## 24.2 RTMP push destinations disappear after reboot

This was fixed for good in v64.0.78. If you observe it on v64.0.78, your shadow file was probably deleted by a rogue backup script. Restore push-destinations-backup.json from backup — the server will rehydrate on next stream recreation.

## 24.3 High RAM and swap thrash

- Look for dead sources — the ghost-RAM guard reports them in journalctl
- Lower ramdisk auto-size tier
- Reduce DVR retention on high-bitrate streams

## 24.4 Getting help

Email support@tvboxhd.com or call 818-264-8383 (USA). Include your version (Settings → About), a Health page screenshot, and the last 200 lines of journalctl -u streamhub.

# Appendix A — Golden Baseline v64.0.78

This whitepaper is written against KlipKast Broadcaster MAX v64.0.78, the current Golden Baseline. v64.0.78 is the first release where RTMP Push, Custom RTMP, and template-created streams all survive reinstall + reboot simultaneously. No in-place changes are permitted; any fix bumps to v64.0.79 or higher.

# Appendix B — Glossary

**HLS —** Apple HTTP Live Streaming. Segmented video over HTTPS, universally playable.

**RTMP —** Real-Time Messaging Protocol. Used for encoder-to-server ingest and downstream push.

**RTSP —** Real-Time Streaming Protocol. Low-latency, common on IP cameras.

**SRT —** Secure Reliable Transport. Modern low-latency contribution over lossy internet.

**PlayBox —** KlipKast's scheduled playout engine for looped and scheduled content.

**Template —** A reusable stream configuration bundling output, destinations and overlays.

**Golden Baseline —** A frozen version. No in-place edits; any fix bumps the version.

**Ramdisk / tmpfs —** RAM-backed filesystem used for hot HLS segments.