What “remote access via static IP” really requires

1. **Reachability:**
    - The TV must be reachable from the hub (routable static public IP or a site-to-site VPN into the showroom LAN).
    - Correct ports opened on the store firewall (varies by OS tools below).
2. **Correct privilege mode on the TV:**
    - You typically need **Developer Mode** (Android TV, webOS, Tizen, Roku) or an **MDM/enterprise** enrollment (Android/Fire TV).
    - Without this, you generally cannot sideload or control media playback programmatically.
3. **Right packaging:**
    - You don’t “run an executable.” You install the platform’s **app package** (APK/IPK/TPK/ZIP) and start it with the vendor’s SDK tools or MDM.
4. **Security/operations:**
    - Dev modes often **timeout** (e.g., webOS ~50 hours) and must be re-enabled on-device.
    - Certificates, pairing keys, and device profiles must be managed.
    - Brand updates can break side-loaded apps.

Feasibility by platform (and the exact how-to)

1) Android TV / Google TV (also Amazon Fire TV, similar via ADB)

Feasible and the most practical of the “remote access” options.

- **What you need**
    - Physical one-time setup: enable *Developer options → USB debugging → ADB over network* (or use USB once).
    - **Network**: TV must be reachable (VPN or public static with firewall rules).
    - **Ports**: ADB default 5555/TCP.
- **What you can do**
    - Install/update your **APK** remotely; start/stop your app; auto-start on boot; run a background service for playback/telemetry.
- **Core commands (from your hub or a jump host)**

```
adb connect <TV_IP>:5555
adb devices
adb install -r player.apk
adb shell pm grant com.your.app android.permission.WAKE_LOCK
adb shell am start -n com.your.app/.MainActivity
```

  Auto-start: implement a `BOOT_COMPLETED` receiver in the app; optionally set **device owner** (via Android Enterprise/MDM like Esper, Scalefusion) for kiosk/autoplay and to block user interference.
- **Pros:** Strong control, robust APIs, decent tooling, MDM options for scale.

- **Cons:** Requires dev/MDM enrollment per TV; not all brands run Android TV.

Fire TV note: Also ADB-capable; enable *ADB debugging* in Developer Options. Install via `adb install`. Similar management story.

---

2) LG webOS

Possible but operationally fragile for production at scale.

- **What you need**
  - **webOS Dev Mode** app installed on each TV and activated (session typically **expires ~50 hours** → must be renewed on-device).
  - LG developer account and device pairing.
  - **Tools**: `ares-setup-device`, `ares-install`, `ares-launch`.
- **What you can do**
  - Install an **IPK** (web app: HTML/CSS/JS) that uses media playback APIs and a websocket/HTTP client for telemetry.
  - Launch/stop the app remotely while dev mode is active.
- **Typical flow**

```
ares-setup-device --add <tv_name> --host <TV_IP> --port 9922 --username prisoner --pr
ares-install player.ipk --device <tv_name>
ares-launch com.your.app
```

- **Cons:** Dev mode renewal and certificate handling make this brittle; background services are limited. Great for pilots, not ideal for unattended nationwide rollout.

---

3) Samsung Tizen

Technically possible; also operationally heavy.

- **What you need**
  - Enable **Developer Mode** on the TV, create a **device profile** and certificate in **Tizen Studio**.
  - **Tools**: `sdb` (Samsung Debug Bridge), `tizen` CLI, or `pkgcmd`.
- **What you can do**
  - Install a **TPK** (web/native) that uses **AVPlay** APIs for video playback and report telemetry via HTTP/MQTT.
- **Typical flow**

```
sdb connect <TV_IP>:26101
tizen install -n player.tpk -t <device_name>
sdb shell  # limited shell, not a general Linux shell
```

- **Cons:** Certs/profiles per fleet, dev mode persistence varies, upgrades can break provisioning. Better than webOS for some deployments, still complex.

---

4) Roku OS

You can sideload during a “developer mode” session, but it’s not meant for fleet ops.

- **What you need**
  - Enable developer mode via on-remote key sequence; set a password.
  - Upload a ZIP of your channel via the built-in dev web server.
- **Limits**
  - **BrightScript** only; limited background control; dev mode is not intended for permanent production. For large fleets you need a **published channel** and Roku business agreements.

---

5) VIDAA / Panasonic “My Home Screen”

Very limited for remote sideloading.
Usually web-style apps with restricted APIs; no stable remote shell; deployments tend to go via the vendor store or on-site provisioning. Treat as **not feasible** for headless remote installs at scale.

Why a generic “run an .exe” isn’t possible

- TVs don’t run Windows; they run **Android, custom Linux, or specialized firmware**.
- Each platform only executes its **own app package** type (APK/IPK/TPK/etc.) and enforces sandboxing.
- Remote shells (SSH) are **not** exposed on retail firmware.

Security & networking checklist (if you still choose this path)

- **Never expose ADB/web dev ports to the public internet.** Use **site-to-site VPN** from hub → showrooms.
- **Firewall allowlist** specific ports per OS tool (ADB 5555, Tizen 26101, webOS 9922, Roku dev web UI 80/8060 etc.).
- **Rotate keys/certs**, keep per-device credentials, audit logs.
- **Lock to kiosk mode** (Android Enterprise/MDM) to prevent user exit.
- **Auto-recovery**: your app must auto-start on boot, loop content offline, and re-sync when network returns.

Practical recommendation for this project

- For a **heterogeneous TV fleet**, relying on ”remote access with static IP” is **operationally risky** (dev mode churn, certs, brand quirks).
- The most reliable, uniform approach is still:
  - **Raspberry Pi (or Intel NUC) on every TV HDMI**

- **LibreELEC + Kodi** or **Python VLC** agent
- Single, brand-agnostic deployment and a clean, testable control surface.

If you *also* have some Android TVs, you can **add an Android TV app path** for those sites (via ADB/MDM) to reduce hardware, but keep Pi/Kodi as the standard baseline.

If you want to proceed with “remote access” anyway—deployment playbooks

Android TV playbook (most feasible)

1. One-time on each TV: enable Developer options + ADB over network (or enroll into an MDM).
2. Network: ensure VPN; open TCP/5555 from hub → TV.
3. From hub CI runner/jump host:

   ```
   adb connect <ip>:5555 || exit 1
   adb install -r player.apk
   adb shell am start -n com.company.player/.MainActivity
   adb shell settings put global stay_on_while_plugged_in 3
   ```

4. App behavior:
   - Foreground player with `MediaPlayer/ExoPlayer`, auto-loop.
   - Background service for **telemetry** (HTTP/MQTT heartbeat).
   - **Offline cache** with a min of 2–3 videos; auto-resume after reboot.

webOS playbook (pilot only)

- Keep a spreadsheet of dev-mode expiry per device and a weekly on-site renewal rota.
- Package IPK; use `ares-*` CLI to push & launch; telemetry via your backend.
- Expect occasional breakage after TV firmware updates.

Tizen playbook (pilot/limited)

- Create and store **device certs**; automate `sdb connect + tizen install` from your hub.
- App uses **AVPlay**; telemetry via HTTP/MQTT; plan for cert rotation and dev-mode lifecycle.

Bottom line

- **Yes**, “remote access via static IP” is **sometimes** possible, **but only** within each vendor’s dev/MDM constraints and with their **app package**, not a generic executable.
- For **reliable, scalable, and uniform control across brands**, prioritize the **Raspberry Pi + LibreELEC + Kodi** approach as the primary architecture, and treat TV-native remote installs as **opportunistic** for Android/Fire TV sites where ADB/MDM is practical.