Linux Bridge Operator Playbook

Linux Bridge Operator Playbook

Last generated: 2026-02-24 15:26:51 | Commit: `d3cdc7f`

Purpose
Operate, diagnose, and recover linux_bridge remote access across Android clients, agent/relay services, and EVA AI routi

Quick Start
1. Confirm bridge services are healthy.
2. Pair the Android device and verify auth token is present.
3. Run the feature checks in order and capture failures in notes.

Table of Contents

Not Included Yet
- Production hardening for multi-operator tenancy and full secret-rotation automation.

Feature: Service Health Baseline
What it does: Validates core services are reachable before app testing.
Why it matters: Prevents misdiagnosing app/device issues when backend services are down.
Where: Linux host shell (`agent / relay / eva endpoints`)
Steps:
1. Source bridge secrets from ~/.config/bridge/secrets.env.
2. Check agent health endpoint and confirm JSON status ok.
3. Check relay health endpoint and confirm response.
4. If using model-router/eva, check its health endpoint too.
Expected result: All required endpoints respond with success; any failing service is identified before mobile testin
Troubleshooting:
- If agent health fails, restart bridge-agent service.
- If relay health fails, restart bridge-relay service and verify BRIDGE_RELAY_TOKEN.
- If EVA health fails, confirm EVA_BASE_URL target service is up.
Command references:
- `curl -fsS ${BRIDGE_AGENT_URL%/}/healthz`
- `curl -fsS ${BRIDGE_RELAY_URL%/ws}/healthz`
- `systemctl --user status bridge-agent.service bridge-relay.service`

Feature: Pair Device And Verify Token
What it does: Creates device auth token and confirms Android prefs are populated.
Why it matters: Most remote failures come from missing/stale auth_token, relay_token, or wrong URLs.
Where: Android app > Connection (`/pair/init + /pair/complete`)
Steps:
1. Run pairing flow (QR or script) and complete on device.
2. Read shared_prefs via adb run-as and verify agent_url/relay_url/auth_token/relay_token/device_key.

3. Confirm eva_url override state matches deployment model.
Expected result: auth_token and relay_token are non-empty and point to public endpoints for off-LAN use.
Troubleshooting:
- If auth_token empty, rerun pair/init + pair/complete.
- If settings show 10.0.2.2 values on physical tablet, patch prefs with scripts/fix_tablet_adb.sh.
- If run-as fails, verify debug build/app id and USB debugging authorization.
Command references:
- `./scripts/fix_tablet_adb.sh <ADB_SERIAL>`
- `$ADB -s <ADB_SERIAL> shell run-as com.bridge.app cat shared_prefs/bridge-settings.xml`

Feature: Terminal Path Validation
What it does: Verifies terminal execution pipeline from Android to linux agent.
Why it matters: Confirms auth + command path before diagnosing AI mode.
Where: Android app > Terminal (`/terminal/exec`)
Steps:
1. Send safe command from app terminal (e.g., pwd).
2. Confirm output returns and command is not blocked unexpectedly.
3. If blocked, review terminal_allowlist in shared prefs and app policy state.
Expected result: Terminal command output appears and host path matches expected linux machine.
Troubleshooting:
- If blocked, add explicit allowlist prefix for command.
- If no output but command runs, inspect UI render path and logcat warnings.
- If unauthorized, refresh token via pairing flow.
Command references:
- `pwd`
- `$ADB -s <ADB_SERIAL> logcat -d | rg -i 'BridgeApi|terminal|blocked'`

Feature: EVA Reachability And Chat
What it does: Checks both health probe and real chat path for EVA.
Why it matters: Health checks can pass while chat fails if auth/path mismatch exists.
Where: Android app > Connection + AI (`/eva/healthz and /eva/chat`)
Steps:
1. Use Connection -> Test Eva and confirm reachable status.
2. Send a short message in AI tab (e.g., test) and verify response.
3. If health passes but chat fails, capture app logs and verify Authorization header behavior in current build.
Expected result: Test Eva is reachable and AI tab returns assistant text without timeout fallback.
Troubleshooting:
- If chat returns missing bearer token/401, confirm app build includes auth header in BridgeApi.evaChat.
- If compat_timeout_or_network persists, verify upstream EVA_BASE_URL service latency and /chat fallback.
- If only fails on cellular, verify dns/https endpoints not LAN addresses.
Command references:
- `$ADB -s <ADB_SERIAL> logcat -d | rg -i 'BridgeNetDiag|eva|timeout|401|unauthorized'`
- `curl -sS -X POST ${BRIDGE_AGENT_URL%/}/eva/chat -H 'Authorization: Bearer <TOKEN>' -H 'content-type: application/json

Feature: Android Build Install Cycle
What it does: Builds, installs, and launches updated Android app on target device.
Why it matters: Provides deterministic recovery after networking/auth code changes.
Where: Linux host scripts (`android_app debug APK`)
Steps:
1. Build debug APK with local gradle distribution or project script.
2. Install to target serial with -r to preserve app data when appropriate.
3. Launch app and run smoke checks for Connection/Terminal/AI tabs.
Expected result: Debug APK installs successfully and app launches on target device.

Troubleshooting:
- If gradle missing, use bundled gradle in android_app/.gradle-dist.
- If kotlin daemon access errors occur, set writable tmp/gradle homes and retry.
- If install fails with signature mismatch, uninstall once then reinstall debug build.
Command references:
- `./scripts/android_build_debug.sh`
- `./scripts/android_install_debug.sh <ADB_SERIAL>`
- `./scripts/android_launch.sh <ADB_SERIAL> com.bridge.app`

Feature: Home Wi-Fi vs Remote Network Checks
What it does: Validates behavior differences between home LAN and cellular/remote networks.
Why it matters: Bridge frequently appears healthy on LAN but fails remotely due to endpoint/token config drift.
Where: Android system + app (`Wi-Fi on/off test matrix`)
Steps:
1. Test on home Wi-Fi: Agent/Eva/Terminal baseline.
2. Disable Wi-Fi and enable cellular; rerun same checks.
3. Record mismatches and inspect app prefs for LAN-only URLs or blank tokens.
Expected result: Same functional results on remote network as on home network.
Troubleshooting:
- If only remote fails, replace private IP or 10.0.2.2 endpoints with public DNS URLs.
- Confirm TLS URLs use correct domain (tplinkdns vs typo domains).
- Verify router/host forwards for agent/relay/eva ports and certificates.
Command references:
- `$ADB -s <ADB_SERIAL> shell svc wifi disable`
- `$ADB -s <ADB_SERIAL> shell svc data enable`
- `$ADB -s <ADB_SERIAL> shell run-as com.bridge.app cat shared_prefs/bridge-settings.xml`

Feature: Incident Diagnostic Bundle
What it does: Collects one bundle of config, health, and logs for fast triage.
Why it matters: Eliminates fragmented copy-paste errors and speeds root-cause analysis.
Where: Linux host shell (`diagnostic log file`)
Steps:
1. Capture app prefs from device and redact secrets for sharing.
2. Capture filtered logcat after reproducing issue once.
3. Capture agent/relay health and recent service logs to one timestamped file.
Expected result: Single diagnostic artifact contains enough data to identify auth, endpoint, or transport faults.
Troubleshooting:
- If terminal closes when running script, remove set -e and run commands one-by-one.
- If adb unavailable in non-interactive context, use absolute platform-tools path.
- If logs are noisy, tighten rg filter to Bridge tags and endpoint names.
Command references:
- `$ADB -s <ADB_SERIAL> logcat -d | rg -i 'BridgeNetDiag|BridgeApi|eva|healthz|timeout|relay|http'`
- `journalctl --user -u bridge-agent.service -u bridge-relay.service --since '30 min ago'`