RemotiveLabs Toolchain — Field Cheatsheet

Virtualize vehicle networks (CAN / LIN / FlexRay / SOME-IP / Automotive Ethernet) → develop & test ECU software with **no hardware**. Scan top-to-bottom to get a vcar running.

✅ Verified 2026-06-24
CLI 0.21.0 · topology-lib 0.20.0

1 🧭 What RemotiveLabs Is

A toolchain to **virtualize vehicle networks** → dev + test ECU software with **no physical hardware**.

Component	Does
RemotiveBroker	gRPC signal-bus server; every ECU/tool connects; raw CAN/LIN/SOME-IP → named signals
RemotiveTopology	declarative `*.instance.yaml` → runnable `docker-compose` of ECU stubs + broker
Behavioral models	Python "mock ECUs" that react to / emit signals
Record & Replay	capture real bus traffic, replay into a virtual bus
RemotiveStudio	desktop/web GUI: visualize signals, browse DBs, record/playback
RemotiveCloud	hosted brokers, recording storage, signal-DB mgmt, sharing
FMU integration	co-simulate FMUs as ECU nodes

2 📦 Install & First Run

Python **≥ 3.10** required.

```
# pipx — recommended, isolated
pipx install remotivelabs-cli
# upgrade: pipx upgrade remotivelabs-cli
```

```
# apt (Debian/Ubuntu)
curl -fsSL https://packages.remotivelabs.com/apt-repo-signing-key.gpg \
  | sudo gpg --dearmor -o /usr/share/keyrings/remotivelabs-apt.gpg
echo "deb [signed-by=/usr/share/keyrings/remotivelabs-apt.gpg] \
  https://packages.remotivelabs.com remotivelabs-apt main" \
  | sudo tee -a /etc/apt/sources.list.d/remotivelabs.list
sudo apt update && sudo apt install remotivelabs-cli
```

```
# rpm (RedHat/Fedora) — /etc/yum.repos.d/remotivelabs.repo
#  baseurl=.../yum/remotivelabs-yum  gpgcheck=0
sudo dnf install remotivelabs-cli

# Python lib (ECU mocks):
pip install -U remotivelabs-topology
# Verify:
remotive --version && remotive topology version
```

- CLI **auto-pulls topology-engine docker image** on first use → pin via `--topology-image` / `REMOTIVE_TOPOLOGY_IMAGE`
- Prefer **pipx** → avoid drifting `remotive` binaries on PATH
- Needs active trial → `remotive topology subscription status`

3 🧱 Core Concepts

Term	Is
Broker	gRPC signal-bus all tools connect to
Namespace	a bus (e.g. a CAN channel)
`*.instance.yaml`	declarative topology definition
Behavioral model	Python mock ECU
Platform	signal-DB binding (`.dbc/.ldf/.fibex/.arxml`)
Raw recording	captured frames (candump / .csv / .raw) — not replayable alone
Recording-session	raw recording **+ signal-DB binding** → replayable
Restbus	synthesized periodic keep-alive frames (no recording)

4 ⌨️ CLI Command Map

Group	Does
`remotive studio`	launch local Studio GUI
`remotive topology`	generate / validate / inspect topology
`remotive broker`	talk to one broker: record, playback, signals, restbus
`remotive cloud`	manage RemotiveCloud (projects/brokers/recordings/DBs/auth)
`remotive connect`	external-system integrations
`remotive tools`	misc utilities
`remotive mcp-server`	MCP server for topology (**Cursor / VSCode / Intellij**)

5 🏗️ topology — Build & Manage

Cmd	Does
`build`	**PRIMARY** — instance YAMLs → docker-compose + ECU stubs
`generate`	**[DEPRECATED] alias for build** (works, warns)
`clean`	remove build output for current workspace
`prune`	remove cache + build output (`--all` = every workspace)
`show`	print info from a signal-DB file
`validate`	**[exp]** validate topology / instance file
`convert`	**[exp]** convert signal-DB format (CAN/LIN only)
`recording-session`	work with recording sessions
`gateway-mapping`	**[exp]** list gateway signal mapping
`workspace`	manage workspaces
`subscription / start-trial`	manage / start license (**30-day trial**)
`version`	engine + bundled broker version
`split`	**[DEPRECATED]** extract per-channel signal DBs

6 🔌 broker — Talk to the Bus

Cmd	Does
`discover`	find brokers on the network
`signals list/subscribe/publish`	list / live-stream / inject signal values
`signals namespaces / frame-distribution`	list buses / frames on a namespace
`record start / stop`	capture bus traffic to a session
`playback play/pause/seek/repeat/open/close/status/list-files`	replay a recording **session** (`*.recordingsession.yaml` , **not** a raw recording — see 9)
`restbus add/start/update/stop/remove/reset`	periodic frame TX at fixed cycle
`files upload/download/delete/reload-configuration`	push/pull broker configs & recordings
`export influxdb`	dump subscribed signals to InfluxDB line-protocol
`scripting`	LUA scripting on the broker
`license`	view / request broker license

cloud group → `auth · organizations · projects · brokers · recordings · signal-databases · storage · service-accounts · samples · licenses`

7 🔁 Typical Workflow

- **Define** `*.instance.yaml` (buses, ECUs, signal DBs)
- **Build** → `remotive topology build -f <instance.yaml> <out_dir>`
- **Run** → `docker compose -f <out_dir>/docker-compose.yml up -d`
- **Mock ECUs** → Python `BehavioralModel` s react to / emit signals
- **Inspect** → `broker signals subscribe` or `remotive studio`
- **Record / Replay** → `record` → `recording-session create` → `playback` (**create step mandatory**, see 9)
- **(Opt) FMU co-sim** → `FMUBehavioralModel` nodes on the bus
- **(Opt) Android** → boot Cuttlefish AAOS + VHAL↔broker bridge

8 🐍 Python Behavioral Model

Package `remotivelabs-topology` . gRPC client in **separate** pkg `remotivelabs-broker` → `from remotivelabs.broker import BrokerClient, Frame, WriteSignal` .

Import	Provides
`behavioral_model.BehavioralModel, PingRequest, RebootRequest`	core mock-ECU loop + control requests
`cli.behavioral_model.BehavioralModelArgs`	arg parser (`.parse()`); URL via `.url`
`namespaces.can.CanNamespace, GenericNamespace, RestbusConfig`	CAN bus + restbus config
`namespaces.some_ip.SomeIPNamespace, SomeIPEvent`	SOME/IP bus
`namespaces.filters` (`E2eSignalsFilter`)	input filters
`control.ControlClient, ControlRequest`	control-plane client
`testing.frames.capture_frames` (+ `hamcrest` , `retry`)	test helpers (`await_at_most`)
`time.async_ticker.create_ticker`	periodic tickers
`ecu_mock.ECUMock`	higher-level stub (re-exports `LinNamespace` , `ScriptedNamespace`)
`comm / metrics / fmu`	primitives / metrics / FMU co-sim (**needs `pip install fmpy`**)

API: `await start()` · `await stop()` · `await run_forever()` · `is_running()` · `reset_restbuses()`

CRITICAL — `broker_client` is a **required positional** (2nd arg) for both `CanNamespace` & `BehavioralModel` . **No `broker_url=` kwarg.** URL from `args.url` (not `.broker_url`).

```
self._broker = BrokerClient(url=args.url)            # args.url
self._bus    = CanNamespace("<Bus>", self._broker)   # 2nd positional
self._bm     = BehavioralModel("<Ecu>", self._broker, namespaces=
[self._bus])
```

Ctor (0.20.0): `BehavioralModel(name, broker_client, namespaces=None, input_handlers=None, control_handlers=None, input_filters=None, on_change=True)` — `on_change` = change-based vs continuous.

CANNOT → not real-time (~tens of Hz, not µs) · no physical-layer / bus-error / CRC sim · doesn't generate topology · can't bypass the broker · version-coupled to broker/schema.

9 📼 Record & Replay (3 steps!)

NOT a 2-step flow. record output = **raw recording**; playback input = a **recording-session** binding it to a **signal DB**. `playback open` **cannot consume a raw recording**.

```
record → raw recording (candump / .csv / .raw)
   |  remotive topology recording-session create   ← REQUIRED middle step
   |     (binds signal DB via mapping.platform)
   ▼
*.recordingsession.yaml → playback open / play
```

1 — record (out = positional, `--namespace` required; or BYO `candump.log` / `.csv`):

```
remotive broker record start <out> --namespace <bus> --url http://<broker>:50051
remotive broker record stop  <out> --namespace <bus> --url http://<broker>:50051
```

2 — bind (recording-session create):

```
remotive topology recording-session create -f <recording> <out.recordingsession.yaml>
```

```
schema: remotive-recording-session:0.2   # MUST match broker
files:
  - location: ./<recording>              # candump.log / .csv / .raw
    type: recording
    default_device: <Channel>
mapping:
  platform:
    - ./platform/<name>.platform.yaml    # → signal DB
  devices:
    <recording-device>: <Channel>        # device → platform channel
```

The `.platform.yaml` points to the real signal DB via its `database:` field (`.dbc` / `.fibex.xml` / `.ldf` / `.arxml`) — here `ChassisMotion.dbc` .

3 — playback (session PATH; or `REMOTIVE_RECORDING_SESSION_PATH`):

```
remotive broker playback open   <session> --url http://<broker>:50051
remotive broker playback play   <session> --offset 10s
```

Run `signals subscribe` **while it plays** → offset advances even when nothing decodes.

10 ⏱️ Offsets & Preconditions

OFFSET UNITS → suffixes `1.15min` / `10s` / `10000ms` . **A bare number = MICROSECONDS (µs), not ms** — always suffix. `--offset 1000` = 1000 µs.

Secondary playback verbs:

- `repeat <session> --start-offset 0 --end-offset 30s` → loop a window
- `seek <session> --offset 5s` · `pause <session>`
- `status` (all open sessions) · `close <session>`

Preconditions (the part that bites):

- Playback broker must **own the channel** + have the **signal DB**. Single-broker setup works directly.
- **Multi-broker / aggregation** proxy does **not** own the physical channel → build a dedicated playback instance (`local_playback.instance.yaml` , source = recording).
- `mapping.devices` must map recording device → real channel, else **0 signals inject** (offset still advances).
- Plain `type: can` needs the **RemotiveBus docker plugin** → use `settings.can.default_driver: udp` (CAN-over-UDP) to avoid it.
- **Schema must match broker** → broker `1.23.0` wants platform `0.16` + recording-session `0.2` ; `0.17` fails `E004: minor version mismatch` .

11 📡 Restbus vs Playback

	Source	Behavior
Playback	a *recorded* file via a recording-session (panel 9)	replays captured traffic
Restbus	nothing recorded — synthesized live	periodic keep-alive frames at a fixed cycle → makes a bus "look alive"

Restbus CLI verbs → panel 6 (`add/start/update/stop/remove/reset`).

12 🖥️ Studio (GUI)

Launch → `remotive studio` (local instance, opens in browser; verified HTTP 200).

- Browse signal DBs (`.dbc/.ldf/.fibex/.arxml`) visually
- Live signal inspection + plot values over time
- Record + replay with a scrubber
- Publish signal values by hand (poke the bus)
- Manage Cloud recordings/brokers without the CLI

Studio = exploration/demo · CLI = automation/CI/scripting.

13 🤖 Android / Cuttlefish

- **No separate Android/AAOS SDK package** (no `remotivelabs-android` on PyPI; only `remotivelabs-topology` + `remotivelabs-broker`).
- Integration = a **Python behavioral model** bridging **broker signals ↔ Android VHAL** (subscribe broker → write VHAL; read VHAL → publish broker).
- RemotiveLabs publishes a Cuttlefish AAOS image `remotivelabs/remotivelabs-cuttlefish` (confirmed on Docker Hub) → boot a virtual AAOS device alongside the bus.
- VHAL limits (slot counts, ranges, encoding) are **Android/VHAL limits, not RemotiveLabs** — bridge code is project-specific.

14 ⚠️ Gotchas & Versions

Gotchas / pro-tips:

- `generate` deprecated → use `remotive topology build`
- Generated compose defaults broker to `${REMOTIVEBROKER_TAG:-1.24}` (was 1.23) → **pin explicitly**
- CLI drops a `remotive.yaml` **workspace marker** into cwd on use
- **Offsets default to microseconds** → always suffix
- **PATH shadowing** → apt `/usr/bin/remotive` vs pipx `~/.local/bin/remotive` drift; `which -a remotive` , keep in lockstep
- Rollback (stay on pipx) → `pipx install --force remotivelabs-cli==<ver>`

Item	Value
CLI	`remotivelabs-cli 0.21.0` (pipx, py3.10)
Topology engine	`RemotiveTopology 0.29.3` (docker image, pulled on first use)
Bundled broker	`RemotiveBroker 1.24`
Python lib	`remotivelabs-topology 0.20.0` (API introspected)
Subscription	trial — **expires 2026-06-24**