

PostgreSQL, TimescaleDB, and platform DB adapters — comparison

PostgreSQL, TimescaleDB, platform DB adapters — advisory

Field	Value
Author	Anton Afanasyeu
Revision	R0
Creation date	2026-06-20
Last modification date	2026-06-20
Co-authored	Cursor Agent (project assistant)
Severity	medium
State	advisory
Document type	technical
Supersedes	—
Design review	—

Table of contents

1. Executive summary
2. When SPEC implementation can start
3. TimescaleDB fit for AndroidCast
 - 3.1 Where it helps in this project
 - 3.2 Practical adoption pattern
 - 3.3 Risks
4. Platform DB adapter layout
 - 4.1 What ac-platform-db owns (core)
 - 4.2 What dialect repos own
 - 4.3 Effort to maintain two adapters
5. MariaDB vs PostgreSQL comparison
 - 5.1 Other bonuses from PostgreSQL
 - 5.2 Reasons to stay on MariaDB longer
6. PostgreSQL vs PostgreSQL + TimescaleDB
7. Migration effort (MariaDB → PostgreSQL)
8. Verdict and recommended path
 - 8.1 Decision
 - 8.2 Phasing
 - 8.3 What to prototype first (low cost)
9. Changelog

Source: 20260620_postgres_timescale_db_platform.md — section links work in PDF viewers that support internal anchors.

Document type: Technical advisory (not a DR/SPEC)

Context: repos reorganizing SPEC R1 approved; ac-platform-db planned as migration-only repo; production today is **MariaDB on Alpine BE** (INFRA.md) with **SQLite** for local dev.

PDF: 20260620_postgres_timescale_db_platform.pdf · Regenerate: `bash scripts/build-all-docs-pdf.sh`

Related: GRAFANA_vs_others_graphvis_pivot.md · specs/20100612_1_scaling.md

1. Executive summary

Question	Short answer
TimescaleDB worth it?	Yes, selectively — for high-frequency cast/WebRTC telemetry and rollups, not for tickets/users/RBAC.
Split ac-platform-db + dialect adapters?	Moderate effort — clean architecture for SPEC Step 3; do not run two production engines long-term.
Pick one engine forever?	PostgreSQL (+ Timescale on hypertables) is the better long-term bet for analytics, cloud, and schema rigor; MariaDB is fine to keep through alpha while adapters are designed.
Verdict	Design ac-platform-db as dialect-neutral migrations + runner; ship ac-platform-db-postgres as primary target; keep ac-platform-db-mariadb as a transition shim (alpha + parity tests), then retire. Enable Timescale only on stats hypertables in the same Postgres instance.

2. When SPEC implementation can start

Green light once all of the following are true:

Gate	Owner	Notes
SPEC R1 approved	PO	Done
<code>git://f0xx.org/ac/*</code> bare repos exist and ACLs work	PO	Done (per PO)
Monolith android_cast read-only for day-to-day dev	PO	You are setting this — avoids drift while extracts land in ac-*
Step 1 verify: push/pull smoke test to ac/ac-docs, ac/ac-scripts	Agent/PO	5-minute git check
Write path agreed: feature branches on ac-* repos, not monolith	Team	After R/O freeze

Recommended first implementation slices (SPEC §9 order):

- Step 2** — push ac-docs, ac-scripts, ac-platform-web from monolith history (no runtime coupling).
 - Step 3** — extract ac-platform-php + ac-platform-db skeleton (this document informs dialect choice).
 - Defer** domain MS splits (Step 6+) until platform Composer packages publish and BE can composer require them.
- Monolith can stay **deployable** for production until Step 8+ per SPEC; R/O on the legacy repo does **not** block new work on ac-* repos.

3. TimescaleDB fit for AndroidCast

TimescaleDB is a **PostgreSQL extension** (Apache-2.0 core; some features under Timescale License). It adds:

- Hypertables** — automatic time partitioning (chunks) for fast range scans
- Compression / columnstore** — large retention with less disk
- `time_bucket()` — idiomatic rollups (hourly/daily session stats)
- Continuous aggregates** — incremental materialized rollups (better than plain PG materialized views for append-heavy series)
- Retention policies** — drop or tier old chunks

3.1 Where it helps in this project

Workload	Current shape	Timescale benefit
Cast session graphs (graph_sessions)	One row per session; dashboards aggregate by day	Low today — row count is modest; MariaDB indexes suffice
Live cast heartbeats / WebRTC stats (growing)	Sub-minute samples per peer/session	High — append-heavy, range + GROUP BY time_bucket
Remote access events	Event log, time-ordered	Medium — compression + retention if volume grows
Tickets, users, RBAC, reports	OLTP	None — keep normal relational tables

Founder pitch (accurate): the win is **fast analytical queries over large time-series** with **compression** and **incremental aggregates**, not replacing your OLTP database.

3.2 Practical adoption pattern

Use **one PostgreSQL instance**:

- Regular tables** — identity, tickets, devices, reports (standard Postgres)

- **Hypertables** — e.g. `cast_stats_samples` (time, session_id, metric, value) after live/WebRTC stats land at scale
- **Continuous aggregates** — e.g. hourly avg_kbps, packet_loss per company for graphs UI

Do **not** run a separate Timescale-only cluster until traffic proves it; extension install on the primary BE Postgres is enough for alpha/beta scale.

3.3 Risks

Risk	Mitigation
Extension not on Alpine/Alpine PHP image path	Use official timescale/timescaledb-ha Docker on be-data-1 or managed Tiger Cloud later
License on advanced features	Core hypertables/compression sufficient; read Timescale licensing before depending on TSL features
PO single-engine ops	Same backup/replica story as Postgres; not a second DB product

4. Platform DB adapter layout

Proposed repos (extends SPEC §6 P1):

```
ac-platform-db           # Contracts: MigrationInterface, SchemaVersion, portable types
ac-platform-db-postgres  # Postgres + Timescale DDL, pgsql migrations, pg_dump helpers
ac-platform-db-mariadb   # MariaDB DDL (transition), mysql migrations
ac-platform-db-sqlite     # Optional: dev-only adapter (today's sqlite path)
```

4.1 What ac-platform-db owns (core)

- Migration runner (ordered files, schema_migrations table)
- **Portable** migration IDs and naming (001_users, not engine-specific paths only)
- Dialect hooks: boolean, json, enum, autoIncrement, timestampMs, upsert
- **Duplication wire** (switching engines): export/import interface

4.2 What dialect repos own

Concern	MariaDB adapter	Postgres adapter
DDL syntax	AUTO_INCREMENT, ENGINE=InnoDB, ENUM(...)	SERIAL/GENERATED, native ENUM or CHECK
Upsert	ON DUPLICATE KEY UPDATE	ON CONFLICT ... DO UPDATE
Migrations	.mariadb.sql or PHP builder	.pgsql.sql + optional CREATE EXTENSION timescaledb
Index DDL	CREATE INDEX IF NOT EXISTS (10.1.1+)	CREATE INDEX CONCURRENTLY for prod
Time helpers	FROM_UNIXTIME(ms/1000)	to_timestamp(ms/1000.0)

4.3 Effort to maintain two adapters

Aspect	Assessment
Mechanical	Medium — you already branch on sqlite vs mysql in Database.php, GraphRepository.php, LiveCastRepository.php
Ongoing cost	High if 100% parity forever — every migration written twice + CI matrix
Sustainable pattern	Single source of truth migrations in PHP/DSL that emit dialect SQL, *or* one canonical Postgres migration + MariaDB shim only until cutover
Duplication while switching	Hard across engines — use one-time pgloader / custom ETL + dual-write window (days), not perpetual sync

Recommendation: treat MariaDB adapter as **legacy bridge**; Postgres adapter as **canonical** after cutover date. CI runs integration tests on Postgres; MariaDB optional until retired.

5. MariaDB vs PostgreSQL comparison

Scores are for **AndroidCast** (PHP PDO microservices, session OLTP, growing stats, VM-first → managed cloud).

Criterion	MariaDB	PostgreSQL	Notes for us
Maintainability (ops)	●●●■	●●●●	MariaDB already on Alpine BE; Postgres is one apk/Docker change
Maintainability (schema)	●●●■	●●●●	Less ENUM/engine-specific drift; stricter types help multi-repo migrations
App code (PHP PDO)	●●●●	●●●●	Both first-class; rewrite mostly SQL strings, not driver
Flexibility (SQL/features)	●●●■	●●●●	PG: richer JSON, window functions, CTEs, partial indexes
OLTP query speed	●●●●	●●●●	Comparable at our row counts; tuning matters more than brand

Analytics / time-range	●●■	●●●■	PG better planners on complex aggregates; Timescale tips this further
Vertical scale	●●●●	●●●●	Single BE VM — both fine to 100GB class with tuning
Horizontal scale	●●●■	●●●●	PG: read replicas, logical replication, Citus/Timescale multi-node later; MariaDB Galera/GTID (cluster0) works but ops-heavy
Cloud managed availability	●●●●	●●●●●	RDS/Aurora MySQL vs RDS/Aurora PG/AlloyDB/Timescale Cloud — PG slightly broader for analytics extensions
SQLite dev parity	●●●■	●●●■	Keep ac-platform-db-sqlite for laptop; neither matches prod perfectly
Existing investment	●●●●●	●■■■■	All prod migrations, schema.mariadb.sql, cluster0 GTID docs
Timescale / time-series	●■■■	●●●●●	MariaDB has no equivalent extension in-tree
Licensing / vendor	GPL / BSL nuances	PostgreSQL License	Both acceptable; Timescale has dual license on some features

5.1 Other bonuses from PostgreSQL

- **Same DB for OLTP + stats** with Timescale on selected tables (no Flink/ETL for dashboards)
- **pgvector / AI extensions** — optional later (Tiger Data stack); not alpha scope
- **Logical replication** — easier blue/green DB upgrades than MySQL dump cuts
- **Stricter constraints** — catches migration bugs earlier when ac-platform-db fans out to many MS repos

5.2 Reasons to stay on MariaDB longer

- Production already stable; **alpha must not hinge on DB migration**
- orchestration/sim/cluster0 GTID MariaDB story is documented
- Team familiarity; zero new extension packaging on Alpine until you choose Docker for be-data-1

6. PostgreSQL vs PostgreSQL + TimescaleDB

Layer	Plain PostgreSQL	+ TimescaleDB
OLTP	Full	Unchanged (regular tables)
Session-level graphs (current)	Indexed BIGINT ms timestamps — adequate	Optional; benefit small until row counts > ~1M
High-frequency samples	Partition manually or suffer sequential scans	Hypertables + compression — large win
Dashboard rollups	Materialized views (full refresh)	Continuous aggregates — incremental
Ops	Standard Postgres backup	+ extension version pin; use timescale/timescaledb-ha images
Query speed (simple)	Baseline	Same for non-hypertable tables
Query speed (time aggregates)	Good with indexes	Much better at scale (time_bucket, chunk exclusion)

Rule of thumb: adopt **Postgres** first; add **Timescale** when (a) live/WebRTC stats storage is defined, and (b) you expect **sustained append rate** or **90+ day retention** of sub-session samples.

7. Migration effort (MariaDB → PostgreSQL)

Area	Effort	Examples in repo
Schema DDL	Medium	ENUM, AUTO_INCREMENT, ENGINE=InnoDB, LONGTEXT
Repository SQL	Medium-High	GraphRepository, LiveCastRepository, Database::tableExists introspection
Migrations	Medium	9+ migration files; idempotent ALTER patterns differ
Config / deploy	Low	pdo_pgsql, DSN in config, Docker compose for be-data-1
Data cutover	Medium	pgloader or CSV per table; schedule maintenance window
Tests	Medium	PHPUnit against Postgres in CI

Estimated engineering (order of magnitude): 1–2 weeks focused for schema + repos + CI, **not** blocking SPEC Steps 2–5 if Postgres is chosen before ac-platform-db migrations are copied out of the monolith.

Duplication window: dual-write (application writes both) for 24–72h *or* read-only monolith + one-shot migration — dual-write across MariaDB↔Postgres is **error-prone**; prefer freeze + migrate + verify.

8. Verdict and recommended path

8.1 Decision

1. **Approve PostgreSQL as the long-term platform database** (managed or VM/Docker).
2. **Use TimescaleDB as an extension on that Postgres** for **telemetry hypertables only** — align with cast stats / live session analytics roadmap.
3. **Structure** ac-platform-db as core + ac-platform-db-postgres (primary) + ac-platform-db-mariadb (temporary parity for current BE until cutover).
4. **Do not** invest in keeping two production engines in sync indefinitely; MariaDB adapter exists to **de-risk alpha**, not to double every feature forever.

8.2 Phasing

Phase	Database	Action
Alpha (now)	MariaDB on Alpine	No forced migration; extract ac-platform-db with both adapters sketched
Platform extract (SPEC Step 3)	CI on Postgres + Timescale Docker	New migrations authored Postgres-first ; MariaDB shim generated or hand-ported for prod continuity
Pre-beta	Postgres (+ Timescale) on be-data-1	One cutover; hypertable for stats; retire MariaDB adapter from CI
Cloud (scaling SPEC)	RDS / Aurora PG / Timescale Cloud	Same ac-platform-db-postgres migrations as job

8.3 What to prototype first (low cost)

1. Docker Compose: Postgres 16 + Timescale — import anonymized graph_sessions + fake 1Hz stats table as hypertable.
2. Re-run graphs dashboard queries with time_bucket vs current MariaDB GROUP BY day — measure at 100k / 1M / 10M rows.
3. Spike ac-platform-db migration interface with one table (schema_migrations) emitted to both dialects.

9. Changelog

Rev	Date	Change
R0	2026-06-20	Initial advisory: Timescale fit, adapter layout, MariaDB vs Postgres, verdict