AEGIRA Technical Build Playbook

AEGIRA Technical Build Playbook

(Time-aware longitudinal workplace risk infrastructure — build-from-scratch guide for an engineer)

---

0) North Star: what we're building

AEGIRA is an enterprise system that continuously captures **worker check-ins** + **context (shift/roster/leave)** and turns them into **time-aware longitudinal signals** to detect emerging risk early. It must be:

- **Time-correct** (event-time first, not "when we received it")
- **Longitudinal** (per-person baselines, trends, deviations)
- **Auditable** (no silent overwrites; every flag explainable)
- **Enterprise-ready** (RBAC, privacy, multi-tenant, scale)

---

1) Users and workflows

Users

- **Worker**: completes check-ins; can see own status, reminders, and "support options."
- **Supervisor / Leading Hand**: sees *actionable* flags (not raw psych data) for their team.
- **HSE / WHS**: sees site-level risk, trends, episodes, outcomes.
- **RTW / Rehab / Case Manager**: manages risk episodes/cases, actions, follow-ups.
- **Org Admin**: config, roster integration, forms, permissions, retention/deletion.

Core workflow (happy path)

1. Worker is assigned to site/team and has a shift schedule.
2. System generates a **CheckInPrompt** for a shift window (pre-start / mid / end).
3. Worker submits **CheckInResponse** (event_time, answers, optional notes).
4. System computes **features** vs personal baseline (deviation, trend, missingness).
5. Risk engine creates/updates a **RiskEpisode** and may open a **Case**.
6. Case gets routed to the right queue (Supervisor / HSE / RTW) with recommended actions.

7. Actions/outcomes logged; model/rule versions recorded; reporting updates.

Edge-case workflow (critical)

- Worker **misses** a check-in → system records a “missed” event at the end of the window.
- Worker submits **late** → system records the late response as its own event and **reconciles** the missed event (without deleting history).
- Worker is **off-shift / on leave** → no missed event should be generated; or if already generated, it must be corrected with an auditable “status override” event.

---

2) Architecture: components and responsibilities

Recommended architecture (pragmatic, production-ready)

Frontend

- Worker app (mobile/web)
- Admin + supervisor dashboard (web)

API & Auth

- API Gateway
- Auth (OIDC/SAML for enterprise, passwordless for workers if needed)
- RBAC enforcement at API layer

Event ingestion (event-time first)

- Events API (append-only)
- Event bus/stream (or queue) for async processing
- Idempotency + dedupe

Storage

- **OLTP DB** (entities, prompts, cases, permissions) – Postgres
- **Event store** (append-only log of check-ins/status changes) – Postgres (append-only table) or Kafka + compacted storage

- **Analytics warehouse** (dashboards, cohort reporting) – BigQuery/Snowflake/Redshift (optional early)

Compute

- Feature engine (batch + streaming)
- Baseline engine (per-person)
- Risk scoring engine (rules now, ML-ready later)

Case management

- Episode builder
- Routing + notifications (email/SMS/Teams/Slack)
- Action/outcome logging

Observability + audit

- Central logging + traces + metrics
- Audit export pipeline (raw + derived + explanation)

Text diagram

Worker App → API Gateway/Auth → Events API → Event Store
Event Store → Stream Processor → Feature/Baseline Store → Risk Engine → Episodes/Cases DB → Dashboards/Notifications
OLTP DB provides Workers/Teams/Rosters/Forms/Permissions to all services.

---

3) Data principles (this is what makes it "time-aware")

Two timestamps everywhere

- **event_time**: when it *happened* (worker submitted, shift started/ended, leave applied)
- **ingested_at**: when AEGIRA *received* it

All analytics, baselines, and missed-check logic use **event_time**.

Immutable event log + "state projections"

- **Event log is append-only** (never overwrite reality).

- Current “truth” is derived via **projections**:
    - “Latest check-in for prompt X”
    - “Worker availability status on date/time”
    - “Current risk episode state”

Idempotency

Every event from a client/integration must include:

- `event_id` (UUID) + `source` + `source_event_id` (optional)
- A dedupe key: `(tenant_id, event_id)` or `(tenant_id, source, source_event_id)`

---

4) Core domain model (entities)

Org structure

- **Tenant**: enterprise customer
- **Site**
- **Team**
- **Worker**: belongs to tenant; mapped to site/team; has privacy attributes
- **Roster/Shift**: shift windows in a timezone, source = manual or integration
- **Leave/OffShift**: explicit unavailability

Check-in system

- **Form**: versioned questions (fatigue, stress, pain, sleep, confidence, etc.)
- **CheckInPrompt**: a scheduled request tied to a worker and a shift window
- **CheckInResponse**: a submission tied to a prompt (or “ad hoc”)

Risk & workflow

- **RiskSignal**: computed result for a specific response/window (deviation metrics + explainability)
- **RiskEpisode**: a time-bounded cluster of risk signals (per worker)
- **Case**: human workflow object (owner, status, actions, outcomes)

Audit and explainability

- **DecisionLog**: records risk engine decisions, rule/model version, feature snapshot hashes

---

5) Event schemas (append-only) — the “source of truth”

You will store every event in a single table/stream, then project.

Base event envelope (every event uses this)

```
{
  "event_id": "uuid",
  "tenant_id": "uuid",
  "event_type": "checkin.response_submitted",
  "event_time": "2026-02-12T07:12:03+11:00",
  "ingested_at": "2026-02-12T07:12:05+11:00",
  "actor": { "type": "worker", "id": "worker_uuid" },
  "source": "worker_app",
  "source_event_id": "optional",
  "schema_version": 1,
  "payload": {}
}
```

Key event types (minimum viable)

1) `roster.shift_upserted`

```
{
  "shift_id": "uuid",
  "worker_id": "uuid",
  "site_id": "uuid",
  "timezone": "Australia/Perth",
  "start_time": "2026-02-12T06:00:00+08:00",
  "end_time": "2026-02-12T18:00:00+08:00",
  "source": "integration|manual"
}
```

2) `worker.availability_set`

(off-shift/leave/sick)

```
{
  "worker_id": "uuid",
  "status": "on_leave|sick|off_shift|available",
  "effective_from": "2026-02-12T00:00:00+11:00",
  "effective_to": "2026-02-14T23:59:59+11:00",
  "reason": "Annual leave",
  "source": "manual|integration"
}
```

3) `checkin.prompt_scheduled`

```
{
  "prompt_id": "uuid",
  "worker_id": "uuid",
  "shift_id": "uuid",
  "form_id": "uuid",
  "window": {
    "opens_at": "2026-02-12T05:15:00+08:00",
    "closes_at": "2026-02-12T06:05:00+08:00"
  },
  "prompt_kind": "pre_shift|mid_shift|post_shift"
}
```

4) `checkin.response_submitted`

```
{
  "response_id": "uuid",
  "prompt_id": "uuid",
  "worker_id": "uuid",
  "submitted_at": "2026-02-12T06:12:03+08:00",
  "answers": {
    "fatigue": 7,
    "stress": 5,
    "pain": 3,
    "sleep_quality": 4,
    "readiness": 5
  },
  "note": "Shoulder a bit tight",
  "client_metadata": { "app_version": "1.0.9" }
}
```

5) `checkin.window_missed`

(system-generated at window close)

```
{
  "missed_id": "uuid",
  "prompt_id": "uuid",
  "worker_id": "uuid",
  "window_closed_at": "2026-02-12T06:05:00+08:00",
  "reason": "no_submission",
  "availability_snapshot": "available|off_shift|on_leave"
}
```

6) `risk.signal_computed`

```
{
  "signal_id": "uuid",
  "worker_id": "uuid",
  "response_id": "uuid",
  "computed_for_time": "2026-02-12T06:12:03+08:00",
  "features": {
    "fatigue_z": 2.1,
    "stress_z": 1.2,
    "pain_delta": 2,
    "missingness_7d": 0.33,
    "volatility_14d": 1.7
  },
  "explanations": [
    { "feature": "fatigue_z", "why": "fatigue 7 vs baseline 3.8" },
    { "feature": "missingness_7d", "why": "2 missed of last 6 prompts" }
  ],
  "engine_version": "ruleset_v3.2"
}
```

7) `risk.episode_updated` / `case.opened` / `case.action_logged`

Keep episodes and case changes as events too (optional in v1; at least log them in DB + DecisionLog).

---

6) The “missed vs late vs overwrite” rule (exactly how to implement)

Golden rule

Never overwrite raw events. Use reconciliation via projections.

What happens when a check-in is missed?

- At `window.closes_at`, the system emits `checkin.window_missed` (append-only).

What happens when a worker submits late?

- The worker app submits `checkin.response_submitted` with `submitted_at` (event_time).
- The projection layer marks:
  - Prompt status = `completed_late`
  - Links response to prompt
  - Keeps the missed event in history but sets prompt’s *current state* to “completed_late”

If a worker was actually off-shift

- An availability event `worker.availability_set` arrives (or shift is removed).
- Projection re-evaluates prompts in that time window:
  - Prompt current state becomes `excused`
  - Any existing `window_missed` remains in event log but is now “explained/excused” in the current state.

Result: You retain a complete audit trail while the “truth” used for metrics is correct.

---

7) Storage design (tables you actually build)

A) OLTP tables (Postgres)

- `tenants, sites, teams, workers`
- `forms, form_versions, questions`
- `shifts` (optional if not purely event-derived)
- `prompts` (current state + references)
- `cases, case_actions, case_outcomes`
- `rbac_roles, rbac_permissions, user_accounts`

B) Event store table (append-only)

`events`

- `event_id (pk)`
- `tenant_id`
- `event_type`
- `event_time (timestamptz)`
- `ingested_at (timestamptz)`
- `actor_type, actor_id`
- `source, source_event_id`
- `payload (jsonb)`
- `schema_version`
- Indexes:
  - `(tenant_id, event_time)`
  - `(tenant_id, actor_id, event_time)`
  - `(tenant_id, event_type, event_time)`
  - Dedupe: `(tenant_id, source, source_event_id)` unique when present

C) Projections (materialized views or tables updated by processors)

- `prompt_state`
  - `prompt_id, worker_id, shift_id, opens_at, closes_at`

    - `state: scheduled|completed_on_time|completed_late|missed|excused`
    - `response_id (nullable)`
    - `last_updated_at`
- `worker_daily_state`
    - worker availability timeline snapshots (or query from events efficiently)
- `feature_store`
    - worker_id, computed_at, feature_name, value, window, version
- `risk_episode_state`
    - worker_id, episode_id, status, severity, opened_at, last_signal_at

---

8) API surface (minimum endpoints)

Worker

- `GET /worker/me`
- `GET /worker/prompts?from=&to=`
- `POST /worker/checkins` *(submits response; idempotent)*
- `GET /worker/history` *(optional)*

Admin / Ops

- `POST /admin/forms`
- `POST /admin/shifts:upsert` *(or integration webhook)*
- `POST /admin/availability`
- `GET /admin/workers`
- `GET /admin/prompts`
- `GET /admin/cases`
- `POST /admin/cases/{id}/actions`

System/Integration

- `POST /events` *(generic ingestion endpoint; recommended internally even if you also expose "friendly" endpoints)*
- `GET /exports/raw-events?worker_id=&from=&to=` *(audit export)*

All POSTs idempotent via header `Idempotency-Key`.

---

9) Time-aware computation (baseline + features)

Baseline strategy (v1: robust + explainable)

For each metric (fatigue, stress, pain, sleep, readiness):

- Use a **rolling window** (e.g., last 28 days of valid responses)
- Baseline = **median** (robust to outliers)
- Dispersion = **MAD** (median absolute deviation) or IQR
- Convert current value to deviation score:
    - `z_robust = (x - median) / (1.4826 * MAD + epsilon)`

Features to compute (minimum)

Level & deviation

- `metric_z_robust` per dimension

Trend

- 7d slope (linear regression on last N points)
- 14d slope

Volatility

- rolling standard deviation / MAD over 14d

Missingness / compliance

- missed rate over 7d / 28d
- streak length of misses

Context-aware features

- “post-nightshift penalty” (optional)
- shift length, consecutive shifts, overtime flags (if available)

Under-reporters (“stoic workers”)

Use **change from baseline** and **within-person volatility**, not absolute thresholds.

- A worker always reporting “2/10” can still show risk if:
  - volatility rises
  - misses increase
  - subtle upward drift emerges

---

10) Risk engine (rules first, ML-ready)

Output objects

- `RiskSignal` (per response/window)
- `RiskEpisode` (cluster of signals)
- Optional `Case` (human workflow)

Ruleset v1 (example)

Trigger a **RiskSignal severity**:

- **High**
  - Any `fatigue_z > 2.5` OR `stress_z > 2.5` OR `pain_delta >= 3`
  - OR `2+ dimensions with z > 2.0`
- **Medium**
  - `fatigue_z > 1.8` or trend slope rising + volatility rising
- **Compliance-risk**
  - missed_rate_7d > 0.4 AND worker availability = available

Episode logic

- Open episode if:
  - High signal, or
  - 2 medium within 7 days, or
  - Compliance-risk sustained 2+ windows
- Close episode if:
  - No medium/high signals for 14 days AND compliance normalizes, OR
  - Case resolved + outcome logged

Routing logic (enterprise-friendly)

- Supervisor sees: “Worker needs a check-in / support”
- HSE/RTW sees: reason codes + suggested next step
- Limit raw metric exposure based on role

---

11) Dashboards and metrics (what workplace teams need)

For supervisors

- Team today: who is flagged, who missed, who is excused
- “Next best action” suggestions
- Compliance view (not shaming; operational)

For HSE/RTW

- Active episodes by site/team
- Time-to-intervention
- Episode outcomes (resolved, escalated, injury prevented proxy)

For exec

- Leading indicators: risk trend, compliance trend
- Lagging indicators: incidents/claims integration later

---

12) Data trust, exportability, auditability (non-negotiable)

Every flag must be explainable

Store, for each `risk.signal_computed`:

- Features used + values
- Baseline snapshot identifiers
- Rule/model version
- Human-readable reasons (“fatigue 7 vs baseline 3.8”)

Raw export

Build a deterministic export that includes:

- Raw events (checkins, missed, availability, roster changes)
- Projections (prompt_state)
- Computed signals (features + reasons)
- Episode/case timeline

Reproducibility

Version everything:

- Form versions
- Ruleset versions
- Feature computation versions

If you re-run history, you must be able to reproduce *the same decision* given the same version inputs.

---

13) Security + privacy (enterprise baseline)

RBAC roles (minimum)

- Worker (self only)
- Supervisor (team-level, redacted)
- HSE/WHS (broader, controlled)
- Case Manager (cases + signals)
- Org Admin (config)
- Platform Admin (internal, restricted)

Privacy patterns

- Separate “identifiers” (name, phone) from health-ish responses
- Role-based redaction: supervisors don’t see raw scores unless allowed
- Tenant isolation in every query

Retention and deletion

- Retention policies per tenant
- Delete worker: tombstone identity + remove direct identifiers; keep aggregated stats if permitted

---

14) Step-by-step build plan (milestones + acceptance criteria)

Phase 1 — Foundations: events + identity + RBAC

Build

- Tenant/org model, workers, teams, sites
- Auth + RBAC
- Append-only `events` table + `/events` ingestion endpoint
- Idempotency + dedupe

Acceptance

- You can ingest 1M events without duplicates
- You can retrieve worker event history efficiently by time range

---

Phase 2 — Shift-aware prompting + missed logic

Build

- Shift ingestion (`roster.shift_upserted`)
- Prompt scheduler (creates `checkin.prompt_scheduled`)
- Window close job emits `checkin.window_missed`
- Availability events + excusal projection

Acceptance

- Prompt_state correctly shows: scheduled / missed / completed_on_time / completed_late / excused
- Late submission never deletes missed history

Phase 3 — Baselines + feature engine

Build

- Baseline computation per worker per metric (rolling median + MAD)
- Feature computation pipeline on new responses and missed events
- Feature store tables

Acceptance

- For any response, you can show baseline and deviation in an export
- Backfill works deterministically

Phase 4 — Risk engine + episodes + case workflow

Build

- Ruleset engine
- RiskSignal creation event
- Episode state machine
- Case creation/routing + action logging

Acceptance

- Given test histories, engine opens/closes episodes correctly
- Every case has reasons + feature snapshot references

Phase 5 — Dashboards + reporting + exports

Build

- Supervisor dashboard views

- HSE/RTW dashboard views
- Export endpoints (raw + derived)
- Metrics: compliance, risk trend, episode counts

Acceptance

- Stakeholder can validate flags from raw export alone
- Basic cohort reporting works per site/team/time window

Phase 6 — Enterprise hardening

Build

- SSO/SAML, audit logs, encryption, secrets management
- Monitoring, alerting, rate limits
- Multi-tenant performance testing
- Data retention/deletion jobs

Acceptance

- Pass security review checklist
- Meet agreed SLOs; can scale per tenant

---

15) Testing strategy (what to test so you don’t get destroyed in pilots)

Unit tests

- event-time ordering
- dedupe/idempotency
- baseline math (median/MAD)
- feature correctness
- ruleset triggers

Scenario tests (must-have)

1. Missed then late submission → final state `completed_late`, history includes `missed`

2. Shift removed after missed → final state `excused`, history preserved
3. Availability set to leave → prompts not generated, or excused if already created
4. Multi-timezone (WA vs NSW) → windows correct
5. Backfill re-run with same versions → identical outcomes

---

16) “Exact process” for an engineer starting tomorrow

1. Implement **events ingestion** + append-only store + idempotency.
2. Implement **rosters + availability** events and projections.
3. Implement **prompt scheduling** and **missed window** generation.
4. Implement **check-in submission** and “late vs on-time” resolution in projections.
5. Implement **baseline engine** (rolling median/MAD) + feature pipeline.
6. Implement **risk signals** (rules v1) + decision logging.
7. Implement **episodes + cases** with routing and action/outcome tracking.
8. Implement **exports** that include raw + derived + explanations.
9. Build dashboards with strict RBAC redaction.
10. Harden for enterprise (SSO, audit logs, retention, deletion, scaling).

What we have/what we need

Good — this is exactly the right moment to audit the structure before it hardens.

You currently have **1 row, 20 columns**, and structurally this is a **flat check-in snapshot table**.

That is **not yet a longitudinal, time-aware risk system**.

Below is a CTO-level audit against the AEGIRA Technical Build Playbook.

🔎 What You Currently Have

Columns:

```
id
company_id
person_id
event_id
check_in_date
hours_slept
sleep_quality
stress_level
physical_condition
notes
readiness_score
readiness_level
sleep_score
stress_score
physical_score
created_at
pain_level
pain_location
physical_condition_notes
pain_score
```

This is essentially:

> “One row = one check-in submission”

That’s fine for v0.
It is **not enough** for AEGIRA’s longitudinal risk infrastructure.

🚨 Critical Gaps (Against AEGIRA Architecture)

1️⃣No True Event-Time System

You have:

- `check_in_date`
- `created_at`

You do NOT have:

- `event_time` (true submission time in timezone)
- `ingested_at`
- timezone
- shift reference
- window open/close
- late vs on-time logic

Right now:

- You cannot determine if this was late.
- You cannot determine if a check-in was missed.
- You cannot detect off-shift excusal.
- You cannot handle multi-timezone enterprises.

This is a major architectural gap.

2️⃣No Shift / Prompt Context

There is:

- No `shift_id`
- No `prompt_id`
- No `window_opens_at`
- No `window_closes_at`

- No `prompt_kind` (pre/mid/post shift)

Without that:

You cannot:

- Generate missed events
- Detect non-compliance
- Reconcile late submissions
- Exclude off-shift workers
- Perform time-aware compliance analytics

Right now this is just a survey table.

3️⃣ No Missed Check-In Events

There is no structure for:

- `checkin.window_missed`
- availability snapshot
- excused status
- reconciliation state

This means:

If a worker misses a check-in, nothing is recorded.

That is catastrophic for:

- Compliance trend detection
- Behavioural drift detection
- Stoic under-reporters
- Risk emergence via disengagement

AEGIRA MUST log missed events.

4️⃣ No Event-Sourcing

This table is state-based, not event-based.

You have:

- `event_id` (but unclear what that represents)
- `id` (primary key)

But:

There is no append-only event store.
No schema version.
No actor metadata.
No source.
No immutability guarantees.

If someone edits a check-in row, you lose history.

AEGIRA requires:

Append-only event log + projections.

5️⃣ No Longitudinal Computation Layer

This file stores:

Raw responses + derived scores.

It does NOT store:

- Baseline snapshot
- Rolling median
- MAD
- z-score deviation
- volatility
- trend slope
- missingness rate
- compliance rate
- feature version
- ruleset version

Which means:

You cannot explain why someone is flagged.

And that kills enterprise trust.

6️⃣ No Availability / Off-Shift Tracking

There is no:

- leave event
- sick event
- off-shift status
- effective_from / effective_to

So if a worker is on leave, your system would:

Incorrectly mark them as non-compliant.

This must be event-driven.

7️⃣ No Episode or Case Model

No:

- risk_signal
- risk_episode
- case
- case_action
- outcome

So currently:

This is not a workplace risk system.
It is a check-in storage table.

🟢 What You Did Right

To be fair:

- `person_id`
- `company_id`
- numeric domain fields
- separation of raw inputs and computed scores
- `created_at`

This is a solid starting capture schema.

But it is not yet longitudinal infrastructure.

🔧 What Must Change (Exact Structural Upgrades)

1️⃣ Convert to Event-Based Capture

Instead of storing only rows in `check_ins_rows`,
you need:

A) Append-only `events` table

Add:

```
event_id (uuid)
tenant_id
event_type
event_time (timestamptz)
ingested_at (timestamptz)
actor_type
actor_id
source
schema_version
payload (jsonb)
```

Your current row becomes:

```
event_type = "checkin.response_submitted"
```

Everything else moves into payload.

2️⃣ Add Prompt + Shift Awareness

You must introduce:

shift_id
prompt_id
window_opens_at
window_closes_at
prompt_kind
timezone

Without this you cannot detect missed vs late.

3️⃣ Add Missed Event Generation

At window_closes_at:

If no response exists:

Emit event:

checkin.window_missed

This must be stored as a row.

Never implied. Always recorded.

4️⃣ Separate Raw Inputs from Derived Scores

Currently you store:

- sleep_score
- stress_score
- readiness_score
- physical_score

Question:

Are these raw user inputs?
Or derived?

If derived, they must be:

- Versioned
- Recomputable
- Logged with feature version

Otherwise historical flags cannot be reproduced.

5️⃣ Add Feature Store

You need a table:

worker_id
computed_at
feature_name
feature_value
window
version

Example:

fatigue_z = 2.1
volatility_14d = 1.7
missing_rate_7d = 0.33

Without this, you are not longitudinal.

6️⃣ Add Availability Model

Introduce event:

worker.availability_set
effective_from
effective_to
status

This is mandatory before pilots.

🧠 Most Important Strategic Point

Right now your data structure supports:

> “How did someone feel today?”

AEGIRA must answer:

> “Is this worker deviating from their personal baseline over time in a way that predicts risk?”

That requires:

- Event-time correctness
- Shift context
- Missed detection
- Baseline modelling
- Deviation tracking
- Versioned risk decisions
- Audit exports

You currently have none of those.

That’s not a criticism — that’s just architectural reality.

🎯 Immediate Upgrade Roadmap (Practical)

If I were your CTO today, I would say:

Step 1 (this week):

- Introduce event store.
- Convert check-in submissions into events.
- Add event_time + timezone.

Step 2:

- Add shift ingestion + prompt scheduling.
- Generate missed events.

Step 3:

- Add baseline computation engine.
- Store deviation metrics separately.

Step 4:

- Implement risk signals + episodes.

Only after that are you building AEGIRA.

🔥 Final CTO Verdict

Your CSV is:

✅ A check-in capture table
❌ Not longitudinal
❌ Not time-aware
❌ Not risk infrastructure
❌ Not enterprise-grade
❌ Not audit-ready

But it is a clean starting foundation.

production-ready redesign

Here’s a **production-ready redesign** of your current `check_ins_rows.csv` into a **time-aware, longitudinal, auditable event log CSV** (what AEGIRA actually needs).

You’ll end up with **one primary export**: `aegira_event_log.csv`.
(You *can* add supporting CSVs later, but this one is the non-negotiable foundation.)

---

1) New CSV: `aegira_event_log.csv` (single source of truth)

Why this format

- **Append-only**: no overwrites
- Handles **late check-ins**, **missed windows**, **off-shift/leave**, and **roster changes**
- Keeps both **event_time** and **ingested_at**
- Supports **reproducibility** (schema_version, source, idempotency keys)

Columns (header)

These columns are designed so multiple event types can live in one CSV (unused fields are blank).

```
event_id,tenant_id,worker_id,event_type,event_time,ingested_at,timezone,source,source_event_id,schema_version,
prompt_id,shift_id,prompt_kind,window_opens_at,window_closes_at,submitted_at,is_late,late_by_seconds,
availability_status,availability_effective_from,availability_effective_to,availability_reason,
hours_slept,sleep_quality,stress_level,physical_condition,pain_level,pain_location,readiness_level,
notes,physical_condition_notes,
sleep_score,stress_score,physical_score,pain_score,readiness_score,
dedupe_key,correlation_id
```

What each event_type means

Minimum set you should emit:

- `checkin.response_submitted` (worker submitted a check-in)
- `checkin.window_missed` (system-generated at window close if no submission)
- `worker.availability_set` (leave/off-shift/sick/available)

- `roster.shift_upserted` (shift added/updated; optional early but strongly recommended)

---

2) Exact rules for missed vs late vs off-shift (how this CSV behaves)

Missed check-in

At `window_closes_at`, if no response exists:

- Write **one new row**: `event_type=checkin.window_missed`
- Do **not** "assume" missed in reporting — it must be an actual recorded event

Late check-in

If worker submits after window close:

- Write **one new row**: `event_type=checkin.response_submitted`
- Set `is_late=true`, fill `late_by_seconds`
- The earlier `checkin.window_missed` row stays forever (audit trail)
- Your projections later decide the prompt's current state = `completed_late`

Off-shift / leave

When a worker is not expected to check in:

- Write **availability** event(s): `event_type=worker.availability_set`
- Then missed events can be **excused** in projections (but still remain if they were already emitted)

---

3) Mapping from your current CSV to the new one

Your current columns map like this:

- `company_id` → `tenant_id`
- `person_id` → `worker_id`
- `event_id` → `correlation_id` *(or `source_event_id` if it came from client)*

- `check_in_date` → **NOT enough** on its own; becomes `event_time` **only if you also store a time + timezone**
- `created_at` → `ingested_at` *(or keep as ingested_at if it truly is “received time”)*
- all check-in answers/scores → same names in new CSV

Important change:
You must start capturing a true submission timestamp:

- `submitted_at` (event_time for the response)
- plus `timezone`

---

4) Example rows (what your export will look like)

A) Worker submitted on time

```
event_id,tenant_id,worker_id,event_type,event_time,ingested_at,timezone,source,source_event_id,schema_version,prompt_id,shift_id,prompt_kind,window_opens_at,window_closes_at,submitted_at,is_late,late_by_seconds,availability_status,availability_effective_from,availability_effective_to,availability_reason,hours_slept,sleep_quality,stress_level,physical_condition,pain_level,pain_location,readiness_level,notes,physical_condition_notes,sleep_score,stress_score,physical_score,pain_score,readiness_score,dedupe_key,correlation_id
66ba70e9-698a-49e6-812d-058f555dec1a,f0ee43c6-5b6e-4946-801e-0bb15eef6d26,1d2ef946-2e50-481a-b33f-7d3c4cdf438b,checkin.response_submitted,2026-02-12T14:02:00+11:00,2026-02-12T14:02:00+11:00,Australia/Sydney,worker_app,,1,pr_001,sh_001,pre_shift,2026-02-12T13:50:00+11:00,2026-02-12T14:10:00+11:00,2026-02-12T14:02:00+11:00,false,,available,,,,7,7,5,6,0,,GREEN,,,"85","50","60","100","74",tenant:f0ee43c6|worker:1d2ef946|prompt:pr_001|submitted:2026-02-12T14:02:00+11:00,66ba70e9-698a-49e6-812d-058f555dec1a
```

B) System recorded a missed window

```
event_id,tenant_id,worker_id,event_type,event_time,ingested_at,timezone,source,source_event_id,schema_version,prompt_id,shift_id,prompt_kind,window_opens_at,window_closes_at,submitted_at,is_late,late_by_seconds,availability_status,availability_effective_from,availability_effective_to,availability_reason,hours_slept,sleep_quality,stress_level,physical_condition,pain_level,pain_location,readiness_level,notes,physical_condition_notes,sleep_score,stress_score,physical_score,pain_score,readiness_score,dedupe_key,correlation_id
miss_9001,f0ee43c6-5b6e-4946-801e-0bb15eef6d26,1d2ef946-2e50-481a-b33f-7d3c4cdf438b,checkin.window_missed,2026-02-12T14:10:00+11:00,2026-02-12T14:10:02+11:00,Australia/Sydney,system,,1,pr_002,sh_001,mid_shift,2026-02-12T17:00:00+11:00,2026-02-12T17:10:00+11:00,,,,"available",,,,,"","","","","","","","","","","","","","",tenant:f0ee43c6|worker:1d2ef946|prompt:pr_002|missed:2026-02-12T17:10:00+11:00,pr_002
```

C) Worker marked as on leave (excuses prompts)

```
event_id,tenant_id,worker_id,event_type,event_time,ingested_at,timezone,source,source_event_id,schema_version,prompt_id,shift_id,prompt_kind,window_opens_at,window_closes_at,submitted_at,is_late,late_by_seconds,availability_status,availability_effective_from,availability_effective_to,availability_reason,hours_slept,sleep_quality,stress_level,physical_condition,pain_level,pain_location,readiness_level,notes,physical_condition_notes,sleep_score,stress_score,physical_score,pain_score,readiness_score,dedupe_key,correlation_id
avail_1001,f0ee43c6-5b6e-4946-801e-0bb15eef6d26,1d2ef946-2e50-481a-b33f-7d3c4cdf438b,worker.availability_set,2026-02-12T00:00:00+11:00,2026-02-12T08:01:00+11:00,Australia/Sydney,admin_dashboard,,1,,,,,,,,,on_leave,2026-02-12T00:00:00+11:00,2026-02-14T23:59:59+11:00,Annual leave,,,,,,,,,,,,,,,tenant:f0ee43c6|worker:1d2ef946|avail:2026-02-12,avail_1001
```

5) If you want the “proper” enterprise version (recommended)

You’ll eventually split into **supporting CSVs** for performance and clarity:

- `workers.csv` (worker, team, site)
- `shifts.csv` (shift windows)
- `prompts.csv` (prompt schedule + current state)
- `event_log.csv` (append-only truth)
- `feature_store.csv` (computed longitudinal features)
- `risk_signals.csv` (flags + explanations)
- `cases.csv` + `case_actions.csv`

But you can start with **event_log.csv** immediately and still be correct.

6) The one change your developer must make today

Right now you only have `check_in_date` and `created_at`.

To become time-aware, you must capture:

- `submitted_at` (with timezone)
- `window_opens_at` / `window_closes_at`

- `prompt_id` + `prompt_kind`
- and you must **emit missed rows**

Without those, “longitudinal risk” will always be guesswork.

If you want, I can also generate an updated CSV file (with headers + example rows) you can hand to your engineer as the target export spec—just say “generate the file”.

Check In.csv Example

Check In.csv