Station Availability & SLA Monitoring

Problem & Context

Riders hit **empty** (no bikes) or **full** (no docks) stations, causing failed trips and churn. Ops lacks a live, reliable view and **actionable alerts** to restore availability quickly. We will build a monitoring + alerting system that converts the raw `status` stream into **SLA events**, routes them to the right on-call, and provides a dashboard for resolution and post-mortems.

Goals & KPIs

- **Reduce stockout minutes** per station/day by **30%** within 8 weeks of rollout.
- **MTTR** (mean time to recover) under **15 minutes** at top-200 stations.
- **Uptime SLA:** ≥ **98.5%** of operating hours with **≥1 bike and ≥1 dock** available.
- Guardrails: **False-positive alerts <5%**, **alerts/dispatcher/hour** stays ≤ baseline.

Scope

Users: Bike Ops Leads (triage), Dispatch/Vans (execution), City/Partner PMs (SLA/analytics).

Data sources (SQLite):

- `status(ts, station_id, bikes_available, docks_available)`
- `station(station_id, name, lat, lon, capacity)`
- `trip(trip_id, start_time, start_station_id, end_station_id, end_time)`
- `weather(ts, temp, precip, wind)`

  Out of scope (v1): van routing optimization; rider incentives. (Handled by separate PRDs.)

SLA & Event Semantics

- **Empty Event:** `bikes_available = 0` sustained for **≥ 5 minutes**.
- **Full Event:** `docks_available = 0` sustained for **≥ 5 minutes**.
- **Event end:** condition clears for **≥ 2 minutes**.
- **Severity:**
  - **S1:** ≥20 min **or** rush hour (07–10, 16–19) at capacity ≥ 35.
  - **S2:** 5–20 min.
  - **S3:** <5 min (trend only; no paging).
- **Chronic station:** >6 events/day **or** >60 stockout minutes/day.

Architecture & Implementation Guide (v1)

1) Data contracts & freshness

- **Ingest cadence:** per-minute snapshots of `status`; mark each row with `ingested_at` and **data freshness** flag if lag >3 min.
- **Timezone:** store timestamps in UTC; display in local city time.
- **Idempotency:** de-dupe on `(station_id, ts)` unique key.

2) Derived models (materialized views)

Create three derived tables:

a) `station_minute` — 1-min snapshots (smoothed).

- Smooth by last-observation-carried-forward if a minute is missing (cap at 3 min).
- Columns: `ts_minute, station_id, bikes_avail, docks_avail, capacity, is_fresh`.

```
-- Assuming status.ts is UNIX epoch seconds; adjust if ISO8601
WITH raw AS (
  SELECT
    station_id,
    datetime(ts, 'unixepoch') AS ts_utc,
    bikes_available, docks_available
```

```
  FROM status
),
min_buckets AS (
  SELECT
    station_id,
    strftime('%Y-%m-%d %H:%M:00', ts_utc) AS ts_minute,
    bikes_available, docks_available
  FROM raw
),
filled AS (
  -- Carry forward last known value per station for up to 3 minutes
  SELECT
    station_id,
    ts_minute,
    bikes_available,
    docks_available
  FROM min_buckets
)
SELECT
  f.ts_minute,
  f.station_id,
  COALESCE(f.bikes_available, 0) AS bikes_avail,
  COALESCE(f.docks_available, 0) AS docks_avail,
  s.capacity,
  1 AS is_fresh  -- set to 0 if lag > 3 min in your ingestion job
FROM filled f
JOIN station s USING(station_id);
```

b) `availability_events` — contiguous runs where empty/full holds (≥5 min).

- Event keys: `event_id (uuid)`, `station_id`, `type` ('EMPTY'/'FULL'), `start_ts`, `end_ts`, `duration_min`, `severity`.

```
WITH flags AS (
  SELECT
    station_id, ts_minute,
```

```
    CASE WHEN bikes_avail = 0 THEN 1 ELSE 0 END AS is_empty,
    CASE WHEN docks_avail = 0 THEN 1 ELSE 0 END AS is_full,
    capacity
  FROM station_minute
),
spans AS (
  SELECT
    station_id, ts_minute, is_empty, is_full, capacity,
    -- Start a group whenever the flag changes
    SUM(CASE
          WHEN (is_empty = 1 AND LAG(is_empty,1,0) OVER (PARTITION BY statio
n_id ORDER BY ts_minute)=0)
           OR (is_full = 1 AND LAG(is_full,1,0) OVER (PARTITION BY station_id ORD
ER BY ts_minute)=0)
          THEN 1 ELSE 0 END
    ) OVER (PARTITION BY station_id ORDER BY ts_minute) AS grp
  FROM flags
),
runs AS (
  SELECT
    station_id,
    MIN(ts_minute) AS start_ts,
    MAX(ts_minute) AS end_ts,
    SUM(is_empty) AS empty_minutes,
    SUM(is_full)  AS full_minutes,
    MAX(capacity) AS capacity
  FROM spans
  GROUP BY station_id, grp
),
events AS (
  SELECT
    station_id,
    'EMPTY' AS type,
    start_ts, end_ts,
    empty_minutes AS duration_min,
    CASE
```

```
    WHEN empty_minutes >= 20 OR (strftime('%H', start_ts) IN ('07','08','09','1
6','17','18') AND capacity >= 35) THEN 'S1'
    WHEN empty_minutes >= 5 THEN 'S2'
    ELSE 'S3'
  END AS severity
 FROM runs WHERE empty_minutes >= 1
 UNION ALL
 SELECT
  station_id, 'FULL', start_ts, end_ts,
  full_minutes AS duration_min,
  CASE
    WHEN full_minutes >= 20 OR (strftime('%H', start_ts) IN ('07','08','09','1
6','17','18') AND capacity >= 35) THEN 'S1'
    WHEN full_minutes >= 5 THEN 'S2'
    ELSE 'S3'
  END
 FROM runs WHERE full_minutes >= 1
)
SELECT
 printf('%s-%s-%s', station_id, type, start_ts) AS event_id,
 *
FROM events
WHERE duration_min >= 5;  -- enforce 5-min SLA threshold
```

c) `station_uptime_daily` — daily SLA ledger.

- Columns: `station_id, day, empty_minutes, full_minutes, uptime_pct, events, mttr_min`.

```
WITH e AS (
 SELECT station_id, date(start_ts) AS day, type, duration_min
 FROM availability_events
),
agg AS (
 SELECT
  station_id, day,
  SUM(CASE WHEN type='EMPTY' THEN duration_min ELSE 0 END) AS empt
```

```
y_minutes,
    SUM(CASE WHEN type='FULL'  THEN duration_min ELSE 0 END) AS full_mi
nutes,
    COUNT(*) AS events
  FROM e
  GROUP BY station_id, day
)
SELECT
  a.station_id, a.day,
  a.empty_minutes, a.full_minutes,
  1440 - (a.empty_minutes + a.full_minutes) AS available_minutes,
  CAST( (1440 - (a.empty_minutes + a.full_minutes)) * 1.0 / 1440 AS REAL ) AS
uptime_pct,
  a.events,
  NULL AS mttr_min  -- compute from events by joining to recover timestamps
FROM agg a;
```

3) Alerting service

- **Triggering:** poll `availability_events` every minute; emit new S1/S2 events only.
- **Deduping:** suppress duplicates if an event with same `(station_id, type)` is active.
- **Escalation:**
  - **S1:** Slack `#ops-oncall` + SMS to on-call; create task in Ops queue.
  - **S2:** Slack `#ops-rebalancing`.
  - **S3:** no alerts; dashboard only.
- **Auto-suppress:** maintenance windows (per-station), planned outages, or if a dispatch task for that station exists with ETA <10 min.
- **Payload:** station name, type, duration so far, current bikes/docks, capacity, nearest van ETA (if available), weather snippet.

4) Dashboard

- **Live Map:** pins colored by severity; tooltip shows `bikes/docks/capacity`, active event duration, freshness.

- **Chronic Stations:** sortable table (stockout minutes, event count, MTTR).
- **SLA Ledger:** `station_uptime_daily` with breach badges and notes.
- **Event detail page:** minute-level chart; weather overlay; trip loss estimate for the interval.

5) Data Quality & Reliability

- **Checks:** no negative values; `bikes_avail + docks_avail <= capacity`; drop outliers; missing minute gap alerts.
- **Latency:** UI shows **"Data delayed by X min"** if `is_fresh=0`.
- **Retention:** raw `status` 90 days; aggregates 12 months.

Expected Behavior (System-wide)

1. **Normal ops:** If a station briefly hits 0 bikes for 2 minutes and recovers → **no alert** (below 5-min threshold), event recorded as S3 trend only.
2. **SLA breach:** Station is empty from 08:03–08:20 → create one **EMPTY S1** event at 08:08 (after 5-min confirmation), page on-call; event ends at 08:22 after 2-min recovery.
3. **Chatter suppression:** If readings flap (0 bikes ↔ 1 bike) minute-to-minute, the 2-minute **end** rule prevents rapid open/close cycles.
4. **Maintenance:** If station is flagged `under_maintenance=1`, **suppress alerts** but still log events for analytics.
5. **Data delay:** If `status` feed is late (>3 min), UI displays **stale** badge; alerts pause until fresh data resumes.
6. **Multi-event overlap:** If station is both full and then empty within the same hour, two independent events are logged and routed.
7. **Chronic flag:** A station accumulating >60 stockout minutes in a day is marked **Chronic** and appears at top of Chronic Stations list.
8. **Backfill:** Late data backfill updates aggregates but **does not** retro-page; dashboards recompute metrics idempotently.

GenAI Assist (Prompt Pack for Workshop)

- **SQL synthesis:**

  "You are a SQL analyst on SQLite. Using tables `status`, `station`, *generate a query to compute EMPTY/FULL events sustained ≥5 minutes with start/end times and severity as defined. Use UTC time and window functions compatible with SQLite."*

- **Root-cause notes:**

  "Given event durations and concurrent weather rows (`precip`, `temp`*), propose a short root-cause label (Rain surge / Commute peak / Event nearby) and confidence."*

- **Ops summary:**

  "Summarize yesterday's top 20 chronic stations with recommended ops actions in 4 bullets each."

Experiment & Analysis Plan

Design: Station-level A/B over 4 weeks. Treatment = alerts enabled; Control = dashboard only.

Primary metric: stockout minutes/station/day. **Secondary:** MTTR, uptime_pct, alerts/dispatcher/hour.

Controls: weather, day-of-week, station capacity.

SQL sketch (effect):

```
SELECT
  t.group AS arm, AVG(d.empty_minutes + d.full_minutes) AS stockout_min
FROM station_uptime_daily d
JOIN experiment_assignment t USING(station_id)
WHERE d.day BETWEEN date('now','-28 days') AND date('now','-1 day')
GROUP BY arm;
```

Success criteria: ≥15% reduction (minimum detectable effect powered at 80% with N stations).

Guardrails: Alert volume per dispatcher/hour not ↑ >20% vs baseline.

Risks & Mitigations

- **Alert fatigue:** dynamic thresholds by capacity/time-of-day; weekly review of rules; S3 sent to dashboard only.
- **Data gaps:** carry-forward capped at 3 min; surface stale markers; fail-safe pause on paging.
- **Ops bandwidth:** auto-batch nearby stations into one field task; integrate ETAs to suppress redundant alerts.
- **Seasonality/weather:** severity boosts during storms; compare like-for-like days in analysis.

Trade-offs (Design Choices)

Option	Pros	Cons	When to choose
5-min threshold	Fewer false pages	Misses very short but painful spikes	Limited dispatcher capacity
3-min threshold	Faster response	Noisier alerts	During peak season or after SLA breaches
Per-station dynamic threshold	Tailored	Slightly complex to tune	High-volume networks

Rollout Plan

1. Week 1–2: Stand up `station_minute`, `availability_events`, `station_uptime_daily`; dry-run alerts to Slack sandbox.
2. Week 3: Limited pilot (50 stations, mixed capacity/regions).
3. Week 4–6: City-wide; start A/B; weekly threshold tuning.
4. Week 8: Lock SLA + hand off to Rebalancing & Incentives inputs.

What This Means for the Roadmap

- **Immediate:** reliable visibility + paging reduces rider-visible outages.

- **Next:** feed **chronic stations** into **Demand Forecasting & Rebalancing** and **Weather-Aware Incentives** to **prevent** events, not just respond.
- **Longer-term:** use event history to inform **Network Expansion & Capacity Sizing** (where to add docks).

Engineering Tickets (Ready-to-Build)

- **ETL:** build `station_minute` with freshness flags; idempotent loader.
- **SQL Views:** `availability_events`, `station_uptime_daily`.
- **Alert Worker:** poll new S1/S2 events; dedupe; route via Slack/SMS; maintenance suppressor.
- **Dashboard:** map, chronic list, SLA ledger, event detail chart.
- **DQ Monitors:** nulls, negative values, sum>capacity, stale feed.
- **Configs:** rush-hour windows, severity thresholds, per-station overrides.

If you want, I can generate the Slack alert payload schema and a seed set of synthetic events so you can demo the end-to-end flow in the workshop.