Case Study: Intelligent Alert Escalation & Resolution System

The system begins with a client sending an HTTP POST request to the API Ingestor (Node/Express), which immediately forwards the alert payload to the Alert Processor service. The Alert Processor enriches the request by generating a canonical alert object, including a server-side timestamp, and then executes the Rule Engine—implemented using JavaScript handlers and a rules.json configuration—to determine the alert's derived status, escalation logic, and lifecycle events. Once the alert is fully evaluated, the processor writes both the alert record and its corresponding lifecycle events into MySQL using mysql2, ensuring data integrity with a single database transaction. After a successful commit, the processor publishes an ALERT_UPDATED message to Kafka (via kafkajs) on the alert-events topic. Downstream, Kafka consumers handle observability: the logs-consumer reads alert messages and writes structured lifecycle logs into Loki, enabling Grafana log dashboards, while the metrics-consumer transforms alert activity into Prometheus metrics, powering Grafana metric dashboards. At the same time, Grafana also queries MySQL directly to visualize alert statuses through SQL panels, combining logs, metrics, and database data into unified monitoring dashboards. A separate background auto-close worker periodically scans MySQL for stale alerts, automatically marks them as AUTO_CLOSED, inserts lifecycle events, and later purges old auto-closed records—all done through controlled SQL transactions. The entire system is orchestrated using Docker Compose, which runs all components (Node services, MySQL, Kafka, Loki, Prometheus, Grafana) with persistent data stored in MySQL volumes and observability data flowing through Loki and Prometheus.

Techstack Used:

Nodejs, Sql, Shell, Docker, Prometheus, Grafana, Promtail, Loki, Kafka, Zoekeeper

CASES SCREENSHOTS:

Input format:

$time = (Get-Date).ToString("yyyy-MM-dd HH:mm:ss")

>> Invoke-WebRequest -Uri "http://localhost:4000/process-alert" -Method POST -Headers @{ "Content-Type" = "application/json" } -Body "{""sourceType"":""overspeed"",""driverId"":""TEST-02"",""severity"":""medium"",""timestamp"":""$time"",""metadata"":{""speed"":121}}"

OVERSPEED — Normal alert (OPEN):

```
PS C:\Users\asus\Desktop\alert-platform-full> $time = (Get-Date).ToString("yyyy-MM-dd HH:mm:ss")
>>
>> Invoke-WebRequest -Uri "http://localhost:4000/process-alert" `
>>     -Method POST `
>>     -Headers @{ "Content-Type" = "application/json" } `
>>     -Body "{""sourceType"":""overspeed"",""driverId"":""TEST-01"",""severity"":""medium"",""timestamp"":""$time"",""metadata"":{""speed"":105}}"
>>


StatusCode        : 201
StatusDescription : Created
Content           : {"alertId":"df7831f6-1c26-4bbd-9f12-8326c7dd0b4d","status":"OPEN"}
RawContent        : HTTP/1.1 201 Created
                    Connection: keep-alive
                    Keep-Alive: timeout=5
                    Content-Length: 66
                    Content-Type: application/json; charset=utf-8
                    Date: Mon, 24 Nov 2025 04:01:32 GMT
                    ETag: W/"42-//Rm1xcpcZi3qGY...
Forms             : {}
Headers           : {[Connection, keep-alive], [Keep-Alive, timeout=5], [Content-Length, 66], [Content-Type, application/json; charset=utf-8]...}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : mshtml.HTMLDocumentClass
RawContentLength  : 66
```

```
mysql> SELECT *  FROM alerts;
+--------------------------------------+-----------+-------------+----------+--------+---------------------+----------------+
| alert_id                             | driver_id | source_type | severity | status | timestamp           | metadata       |
+--------------------------------------+-----------+-------------+----------+--------+---------------------+----------------+
| df7831f6-1c26-4bbd-9f12-8326c7dd0b4d | TEST-01   | overspeed   | medium   | OPEN   | 2025-11-24 09:31:32 | {"speed": 105} |
+--------------------------------------+-----------+-------------+----------+--------+---------------------+----------------+
1 row in set (0.00 sec)
```

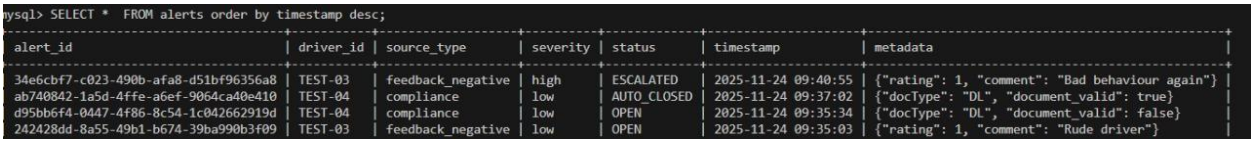
OVERSPEED — Trigger escalation (3 times within window):

```
mysql> SELECT *  FROM alerts order by timestamp desc;
+--------------------------------------+-----------+-------------+----------+-----------+---------------------+----------------+
| alert_id                             | driver_id | source_type | severity | status    | timestamp           | metadata       |
+--------------------------------------+-----------+-------------+----------+-----------+---------------------+----------------+
| 69292914-67a6-41bc-b675-267bd7e6b018 | TEST-02   | overspeed   | critical | ESCALATED | 2025-11-24 09:33:24 | {"speed": 120} |
| 0c4d76a0-0cd8-42c3-be3a-af4a0cdeed6d | TEST-02   | overspeed   | medium   | OPEN      | 2025-11-24 09:33:21 | {"speed": 120} |
| ff2e3727-aa7e-42e1-ad68-d4fd799af9b1 | TEST-02   | overspeed   | medium   | OPEN      | 2025-11-24 09:33:17 | {"speed": 120} |
| df7831f6-1c26-4bbd-9f12-8326c7dd0b4d | TEST-01   | overspeed   | medium   | OPEN      | 2025-11-24 09:31:32 | {"speed": 105} |
+--------------------------------------+-----------+-------------+----------+-----------+---------------------+----------------+
4 rows in set (0.01 sec)
```

NEGATIVE FEEDBACK — Single bad feedback (OPEN):

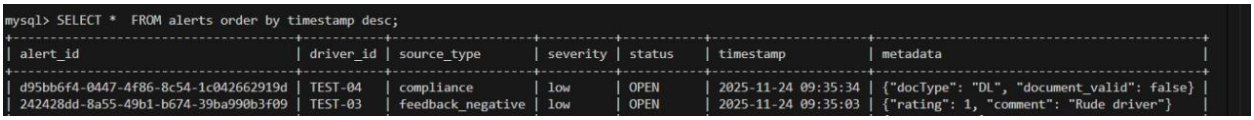
```
mysql> SELECT *  FROM alerts order by timestamp desc;
+--------------------------------------+-----------+-----------------+----------+----------+---------------------+-----------------------------------------+
| alert_id                             | driver_id | source_type     | severity | status   | timestamp           | metadata                                |
+--------------------------------------+-----------+-----------------+----------+----------+---------------------+-----------------------------------------+
| 242428dd-8a55-49b1-b674-39ba990b3f09 | TEST-03   | feedback_negative | low    | OPEN     | 2025-11-24 09:35:03 | {"rating": 1, "comment": "Rude driver"} |
| 69292914-67a6-41bc-b675-267bd7e6b018 | TEST-02   | overspeed       | critical | ESCALATED | 2025-11-24 09:33:24 | {"speed": 120}                          |
| 0c4d76a0-0cd8-42c3-be3a-af4a0cdeed6d | TEST-02   | overspeed       | medium   | OPEN     | 2025-11-24 09:33:21 | {"speed": 120}                          |
| ff2e3727-aa7e-42e1-ad68-d4fd799af9b1 | TEST-02   | overspeed       | medium   | OPEN     | 2025-11-24 09:33:17 | {"speed": 120}                          |
| df7831f6-1c26-4bbd-9f12-8326c7dd0b4d | TEST-01   | overspeed       | medium   | OPEN     | 2025-11-24 09:31:32 | {"speed": 105}                          |
+--------------------------------------+-----------+-----------------+----------+----------+---------------------+-----------------------------------------+
```

NEGATIVE FEEDBACK — Second bad feedback triggers ESCALATION:



```
mysql> SELECT *  FROM alerts order by timestamp desc;
+--------------------------------------+-----------+-----------------+----------+-------------+---------------------+-----------------------------------------+
| alert_id                             | driver_id | source_type     | severity | status      | timestamp           | metadata                                |
+--------------------------------------+-----------+-----------------+----------+-------------+---------------------+-----------------------------------------+
| 34e6cbf7-c023-490b-afa8-d51bf96356a8 | TEST-03   | feedback_negative | high   | ESCALATED   | 2025-11-24 09:40:55 | {"rating": 1, "comment": "Bad behaviour again"} |
| ab740842-1a5d-4ffe-a6ef-9064ca40e410 | TEST-04   | compliance      | low      | AUTO_CLOSED | 2025-11-24 09:37:02 | {"docType": "DL", "document_valid": true}       |
| d95bb6f4-0447-4f86-8c54-1c042662919d | TEST-04   | compliance      | low      | OPEN        | 2025-11-24 09:35:34 | {"docType": "DL", "document_valid": false}      |
| 242428dd-8a55-49b1-b674-39ba990b3f09 | TEST-03   | feedback_negative | low    | OPEN        | 2025-11-24 09:35:03 | {"rating": 1, "comment": "Rude driver"}         |
```

COMPLIANCE — Document expired (OPEN):

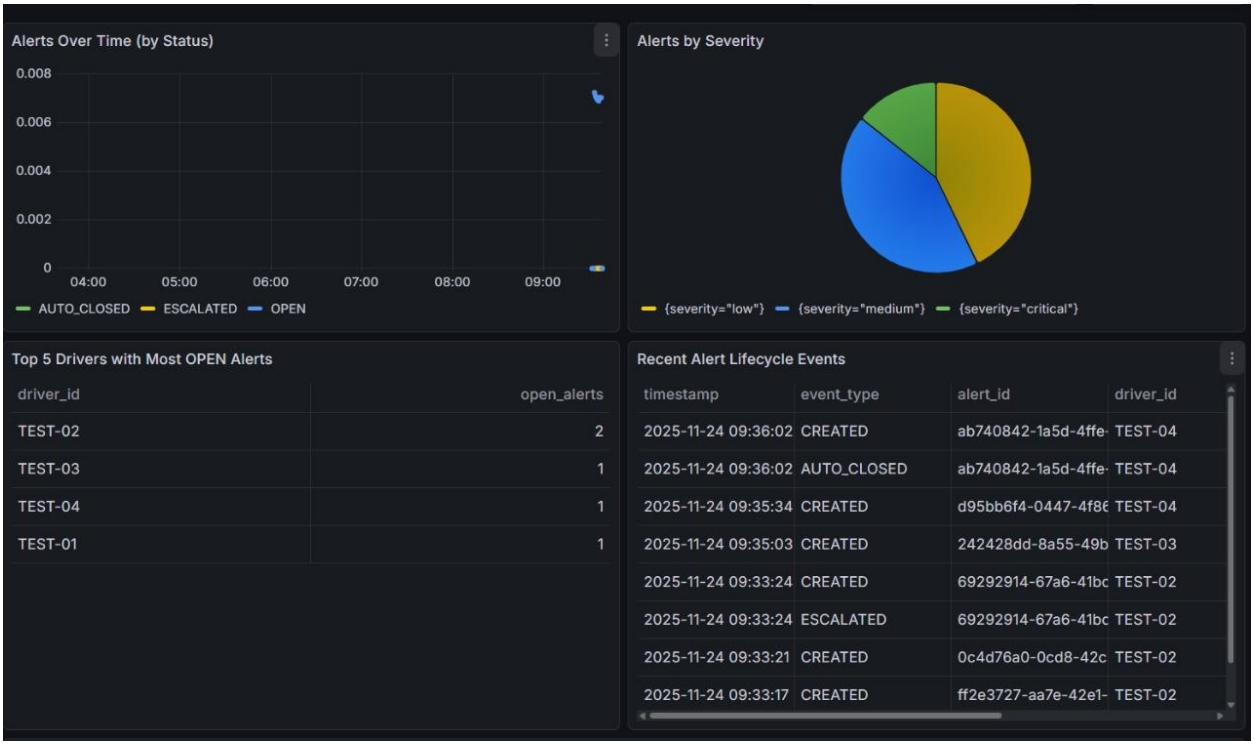


```
mysql> SELECT *  FROM alerts order by timestamp desc;
+--------------------------------------+-----------+-----------------+----------+----------+---------------------+-----------------------------------------+
| alert_id                             | driver_id | source_type     | severity | status   | timestamp           | metadata                                |
+--------------------------------------+-----------+-----------------+----------+----------+---------------------+-----------------------------------------+
| d95bb6f4-0447-4f86-8c54-1c042662919d | TEST-04   | compliance      | low      | OPEN     | 2025-11-24 09:35:34 | {"docType": "DL", "document_valid": false} |
| 242428dd-8a55-49b1-b674-39ba990b3f09 | TEST-03   | feedback_negative | low    | OPEN     | 2025-11-24 09:35:03 | {"rating": 1, "comment": "Rude driver"} |
```

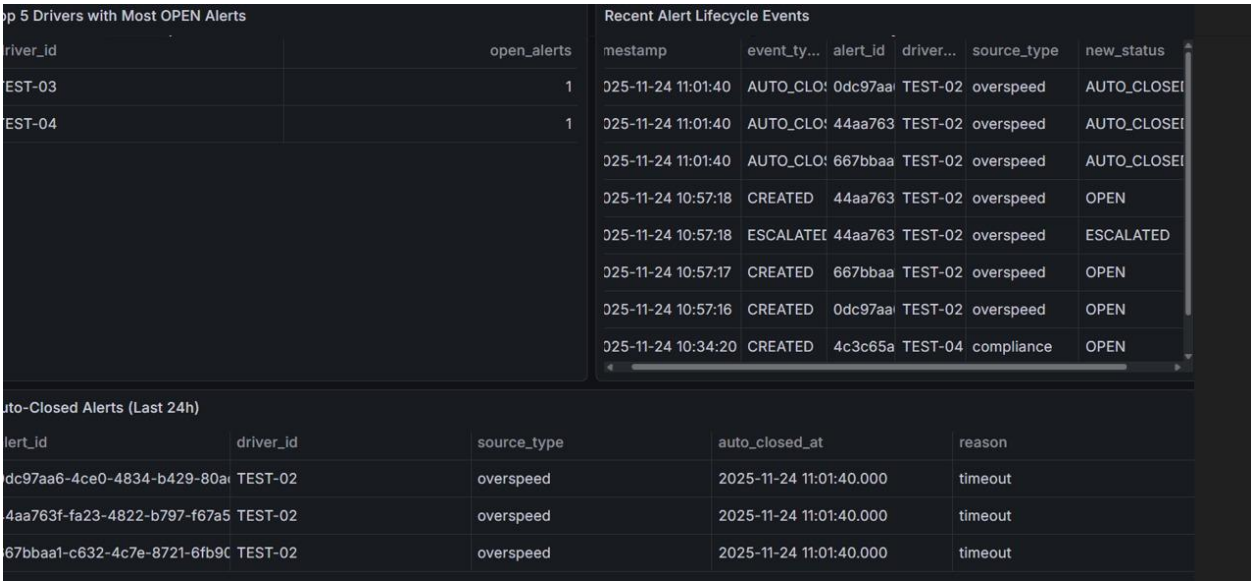
COMPLIANCE — Auto-close because document is valid now:

```
mysql> SELECT *  FROM alerts order by timestamp desc;
+--------------------------------------+-----------+-----------------+----------+-------------+---------------------+-----------------------------------------+
| alert_id                             | driver_id | source_type     | severity | status      | timestamp           | metadata                                |
+--------------------------------------+-----------+-----------------+----------+-------------+---------------------+-----------------------------------------+
| ab740842-1a5d-4ffe-a6ef-9064ca40e410 | TEST-04   | compliance      | low      | AUTO_CLOSED | 2025-11-24 09:37:02 | {"docType": "DL", "document_valid": true}  |
| d95bb6f4-0447-4f86-8c54-1c042662919d | TEST-04   | compliance      | low      | OPEN        | 2025-11-24 09:35:34 | {"docType": "DL", "document_valid": false} |
```

GRAFANA representation with dashboards:



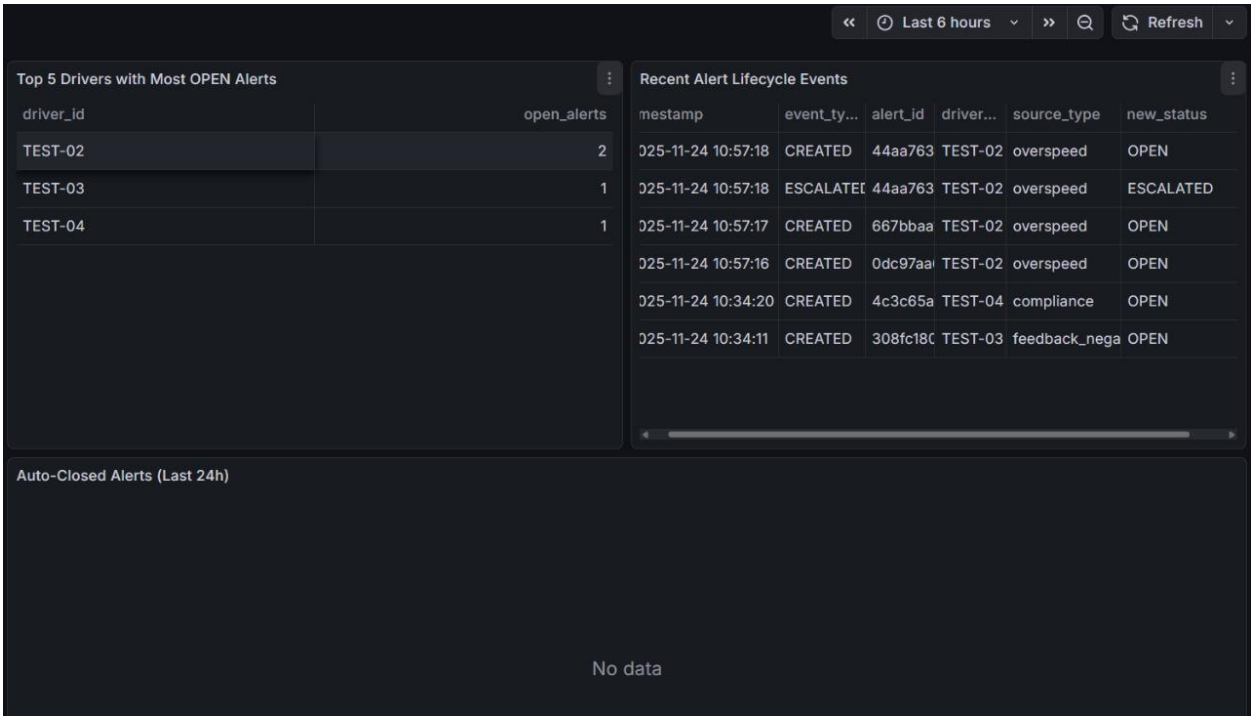
Top Offenders, Alerts by severity, Recent alerts, Trend Over Time



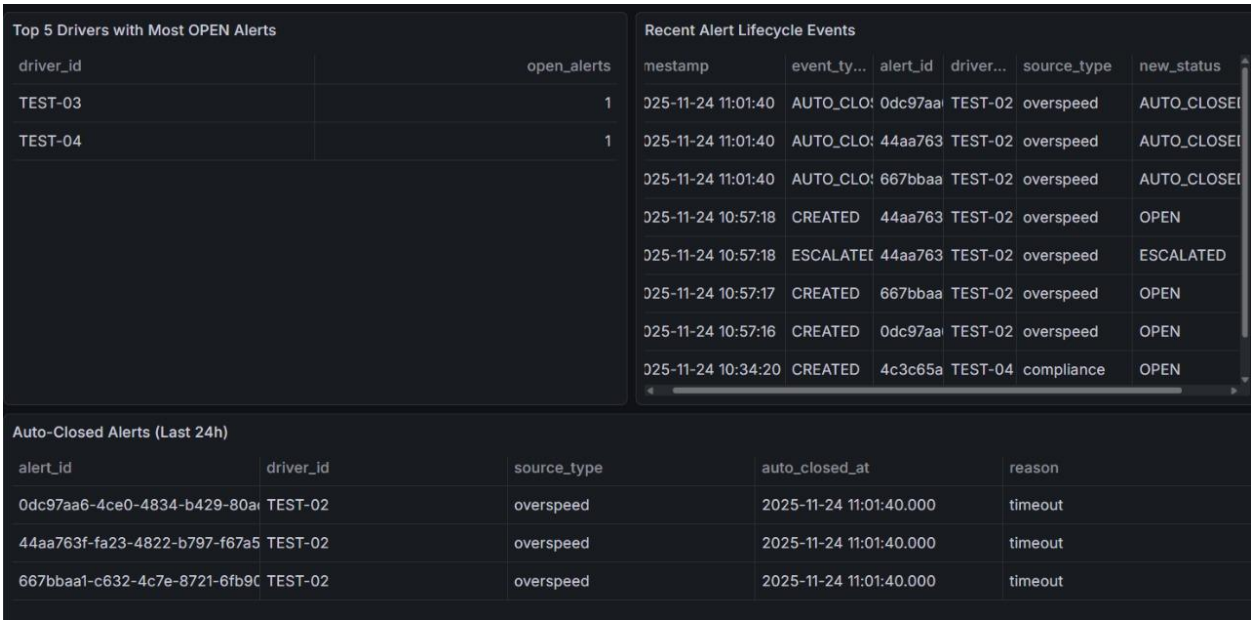
Auto-Closed Alerts Transparency

AUTO-CLOSE worker test:

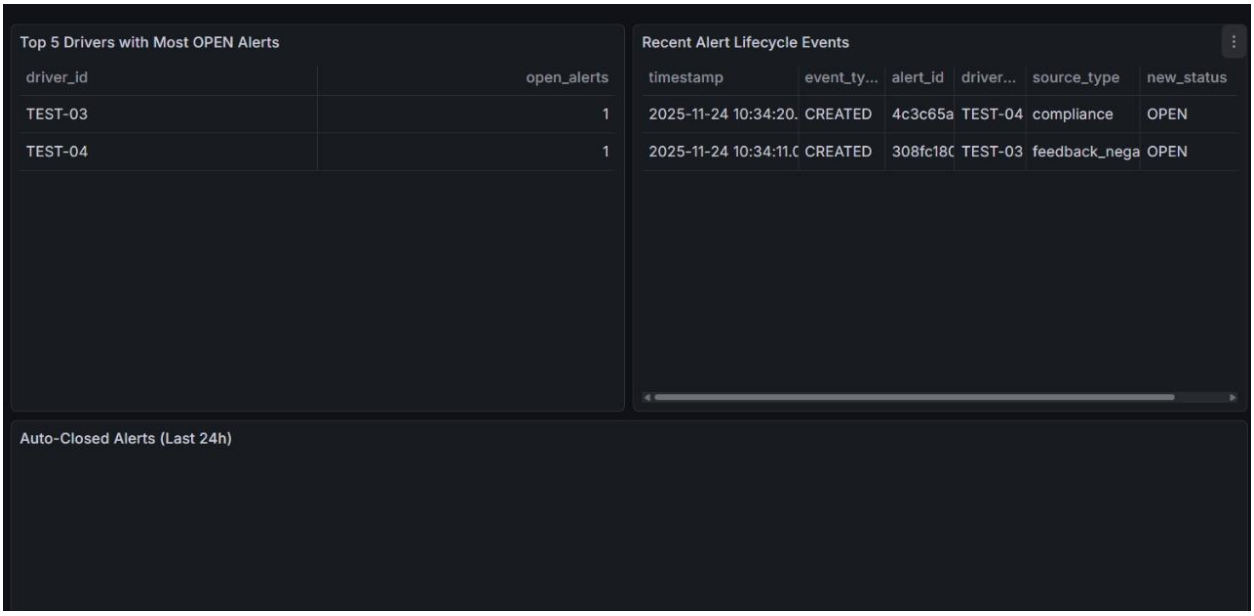
Before autoclose works:



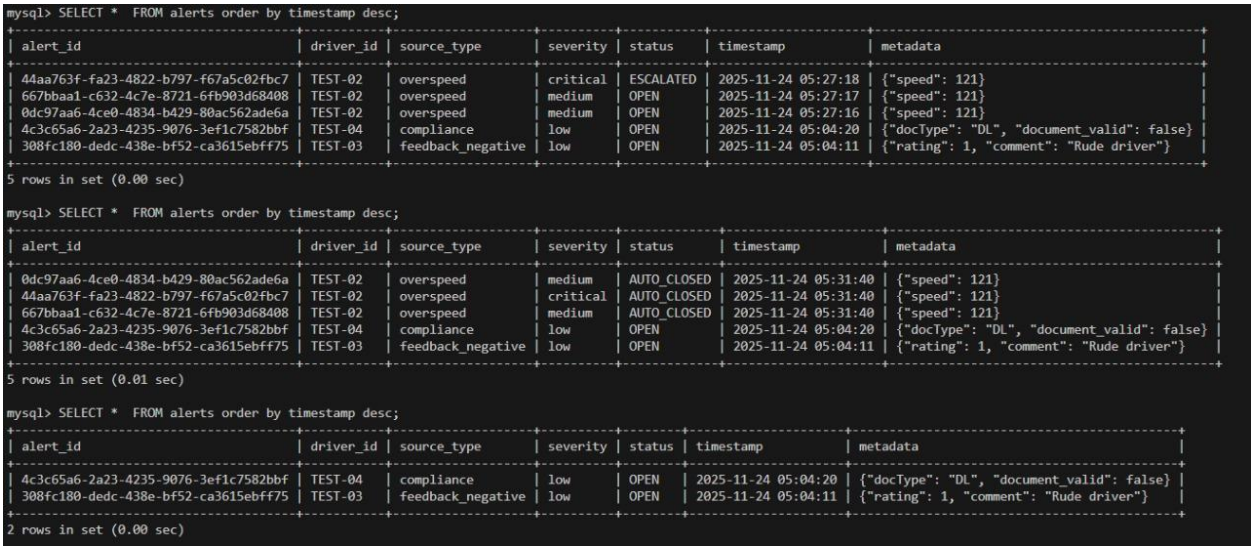
After auto close:



After auto delete:



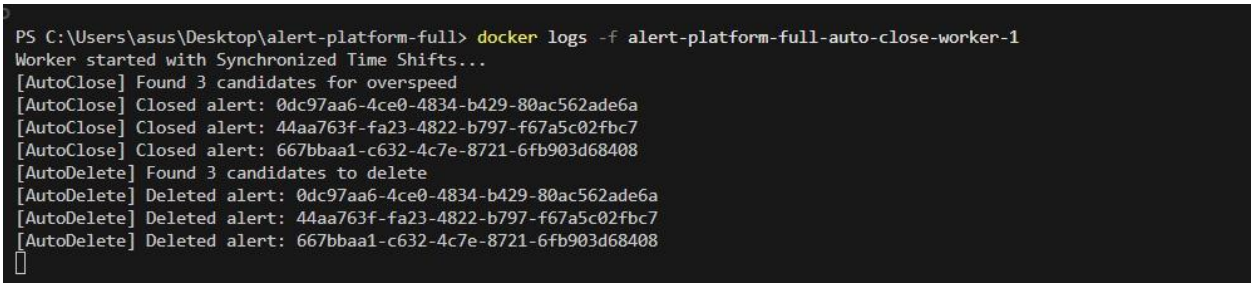
During auto close and autodelete database and auto close worker logs



```
mysql> SELECT *  FROM alerts order by timestamp desc;
+--------------------------------------+-----------+------------------+----------+-----------+---------------------+----------------------------------------------------+
| alert_id                             | driver_id | source_type      | severity | status    | timestamp           | metadata                                           |
+--------------------------------------+-----------+------------------+----------+-----------+---------------------+----------------------------------------------------+
| 44aa763f-fa23-4822-b797-f67a5c02fbc7 | TEST-02   | overspeed        | critical | ESCALATED | 2025-11-24 05:27:18 | {"speed": 121}                                     |
| 667bbaa1-c632-4c7e-8721-6fb903d68408 | TEST-02   | overspeed        | medium   | OPEN      | 2025-11-24 05:27:17 | {"speed": 121}                                     |
| 0dc97aa6-4ce0-4834-b429-80ac562ade6a | TEST-02   | overspeed        | medium   | OPEN      | 2025-11-24 05:27:16 | {"speed": 121}                                     |
| 4c3c65a6-2a23-4235-9076-3ef1c7582bbf | TEST-04   | compliance       | low      | OPEN      | 2025-11-24 05:04:20 | {"docType": "DL", "document_valid": false}         |
| 308fc180-dedc-438e-bf52-ca3615ebff75 | TEST-03   | feedback_negative | low      | OPEN      | 2025-11-24 05:04:11 | {"rating": 1, "comment": "Rude driver"}            |
+--------------------------------------+-----------+------------------+----------+-----------+---------------------+----------------------------------------------------+
5 rows in set (0.00 sec)

mysql> SELECT *  FROM alerts order by timestamp desc;
+--------------------------------------+-----------+------------------+----------+-------------+---------------------+----------------------------------------------------+
| alert_id                             | driver_id | source_type      | severity | status      | timestamp           | metadata                                           |
+--------------------------------------+-----------+------------------+----------+-------------+---------------------+----------------------------------------------------+
| 0dc97aa6-4ce0-4834-b429-80ac562ade6a | TEST-02   | overspeed        | medium   | AUTO_CLOSED | 2025-11-24 05:31:40 | {"speed": 121}                                     |
| 44aa763f-fa23-4822-b797-f67a5c02fbc7 | TEST-02   | overspeed        | critical | AUTO_CLOSED | 2025-11-24 05:31:40 | {"speed": 121}                                     |
| 667bbaa1-c632-4c7e-8721-6fb903d68408 | TEST-02   | overspeed        | medium   | AUTO_CLOSED | 2025-11-24 05:31:40 | {"speed": 121}                                     |
| 4c3c65a6-2a23-4235-9076-3ef1c7582bbf | TEST-04   | compliance       | low      | OPEN        | 2025-11-24 05:04:20 | {"docType": "DL", "document_valid": false}         |
| 308fc180-dedc-438e-bf52-ca3615ebff75 | TEST-03   | feedback_negative | low      | OPEN        | 2025-11-24 05:04:11 | {"rating": 1, "comment": "Rude driver"}            |
+--------------------------------------+-----------+------------------+----------+-------------+---------------------+----------------------------------------------------+
5 rows in set (0.01 sec)

mysql> SELECT *  FROM alerts order by timestamp desc;
+--------------------------------------+-----------+------------------+----------+--------+---------------------+----------------------------------------------------+
| alert_id                             | driver_id | source_type      | severity | status | timestamp           | metadata                                           |
+--------------------------------------+-----------+------------------+----------+--------+---------------------+----------------------------------------------------+
| 4c3c65a6-2a23-4235-9076-3ef1c7582bbf | TEST-04   | compliance       | low      | OPEN   | 2025-11-24 05:04:20 | {"docType": "DL", "document_valid": false}         |
| 308fc180-dedc-438e-bf52-ca3615ebff75 | TEST-03   | feedback_negative | low      | OPEN   | 2025-11-24 05:04:11 | {"rating": 1, "comment": "Rude driver"}            |
+--------------------------------------+-----------+------------------+----------+--------+---------------------+----------------------------------------------------+
2 rows in set (0.00 sec)
```

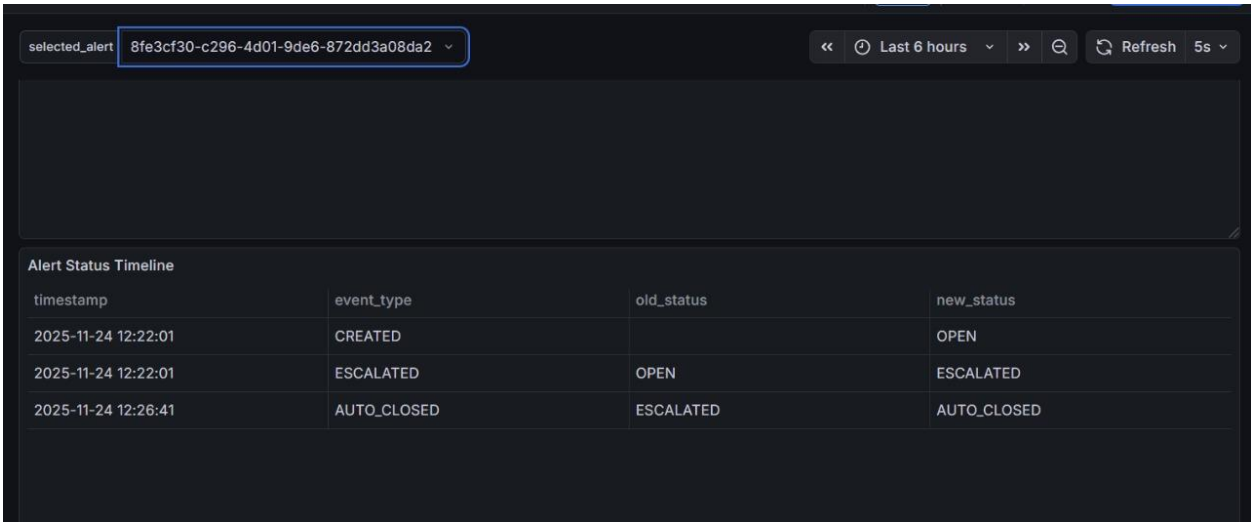


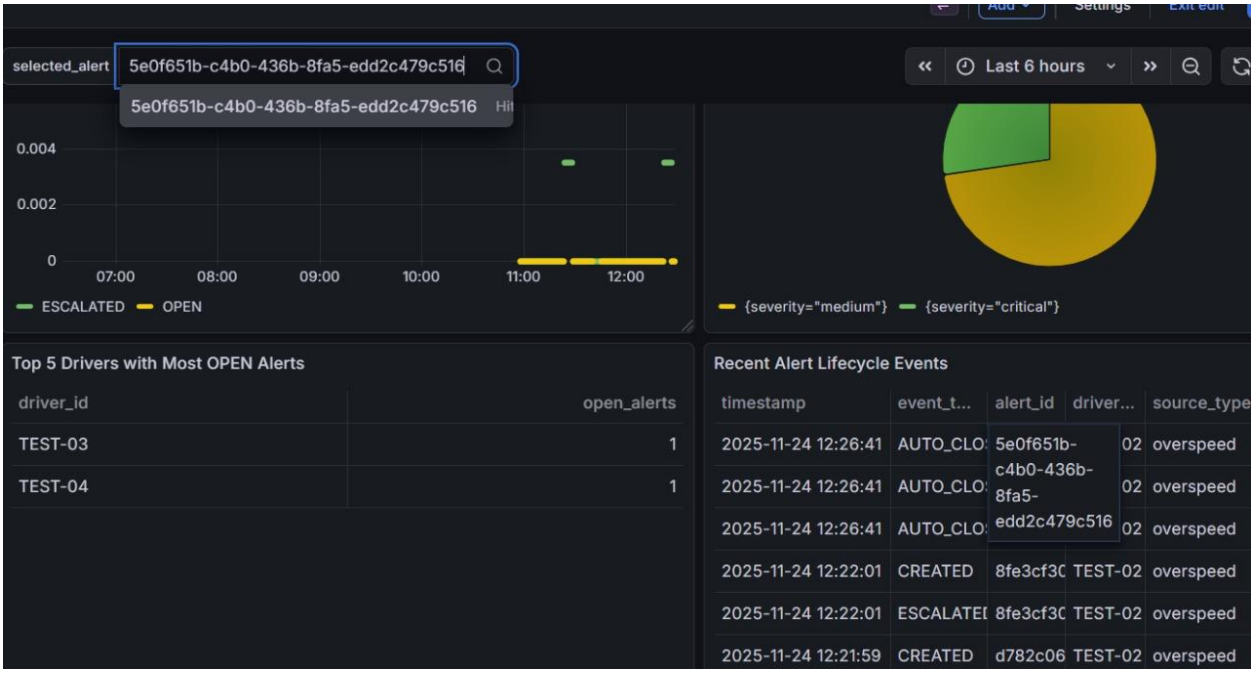
```
PS C:\Users\asus\Desktop\alert-platform-full> docker logs -f alert-platform-full-auto-close-worker-1
Worker started with Synchronized Time Shifts...
[AutoClose] Found 3 candidates for overspeed
[AutoClose] Closed alert: 0dc97aa6-4ce0-4834-b429-80ac562ade6a
[AutoClose] Closed alert: 44aa763f-fa23-4822-b797-f67a5c02fbc7
[AutoClose] Closed alert: 667bbaa1-c632-4c7e-8721-6fb903d68408
[AutoDelete] Found 3 candidates to delete
[AutoDelete] Deleted alert: 0dc97aa6-4ce0-4834-b429-80ac562ade6a
[AutoDelete] Deleted alert: 44aa763f-fa23-4822-b797-f67a5c02fbc7
[AutoDelete] Deleted alert: 667bbaa1-c632-4c7e-8721-6fb903d68408
```

NOTE: Rules used

```
{
  "overspeed": {
    "escalate_if_count": 3,
    "window_mins": 2,
    "auto_close_after_mins": 4
  },
  "feedback_negative": {
    "escalate_if_count": 2,
    "window_mins": 1440,
    "auto_close_after_mins": 2880
  },
  "compliance": {
    "auto_close_if": "document_valid"
  }
}
```

Alert Drill-Down (Alert status timeline):




selected_alert
5e0f651b-c4b0-436b-8fa5-edd2c479c516
5e0f651b-c4b0-436b-8fa5-edd2c479c516
Last 6 hours
0.004
0.002
0
07:00
08:00
09:00
10:00
11:00
12:00
ESCALATED
OPEN
{severity="medium"}
{severity="critical"}
Top 5 Drivers with Most OPEN Alerts
driver_id
open_alerts
TEST-03
1
TEST-04
1
Recent Alert Lifecycle Events
timestamp
event_t...
alert_id
driver...
source_type
2025-11-24 12:26:41
AUTO_CLO
5e0f651b-c4b0-436b-8fa5-edd2c479c516
02
overspeed
2025-11-24 12:26:41
AUTO_CLO
02
overspeed
2025-11-24 12:26:41
AUTO_CLO
02
overspeed
2025-11-24 12:22:01
CREATED
8fe3cf3(
TEST-02
overspeed
2025-11-24 12:22:01
ESCALATEI
8fe3cf3(
TEST-02
overspeed
2025-11-24 12:21:59
CREATED
d782c06
TEST-02
overspeed

DATABASE

```
CREATE TABLE IF NOT EXISTS alerts (
  alert_id VARCHAR(50) PRIMARY KEY,
  driver_id VARCHAR(50),
  source_type VARCHAR(50) NOT NULL,
  severity VARCHAR(20),
  status VARCHAR(20),
  timestamp DATETIME,
  metadata JSON
);

CREATE TABLE IF NOT EXISTS alert_events (
  event_id INT AUTO_INCREMENT PRIMARY KEY,
  alert_id VARCHAR(50),
  event_type VARCHAR(50),
  old_status VARCHAR(20),
  new_status VARCHAR(20),
  timestamp DATETIME,
  metadata JSON,
  INDEX idx_alert_id (alert_id),
  CONSTRAINT fk_alert FOREIGN KEY (alert_id) REFERENCES alerts(alert_id)
    ON DELETE CASCADE
);
```

AUTO CLOSE WORKER(CODE SNIPPET)

```
const db = require('../db');
const rules = require('../config/rules.json');

// Configuration
const POLL_MS = Number(process.env.AUTO_CLOSE_POLL_MS || 5000);
const DELETE_AFTER_MINUTES = 5;

function getShiftedTime(dateObj = new Date()) {
  // 330 minutes = 5 hours 30 minutes
  return new Date(dateObj.getTime() - (330 * 60 * 1000));
}

// ---------------------------------------------------------
// AUTO CLOSE WORKER
// ---------------------------------------------------------
async function autoClose() {
  const conn = await db.getConnection();

  try {
    for (const [src, cfg] of Object.entries(rules)) {
      if (!cfg.auto_close_after_mins) continue;

      const mins = cfg.auto_close_after_mins;

      // 1. Calculate Real Cutoff: Now - 2 mins
      const realCutoff = new Date(Date.now() - (mins * 60 * 1000));
```

```
// 2. Shift it (-5.5h) so we compare "Apples to Apples" with DB
const shiftedCutoff = getShiftedTime(realCutoff);

const [rows] = await conn.execute(
  `SELECT alert_id, status
   FROM alerts
   WHERE source_type = ?
    AND status IN ('OPEN', 'ESCALATED')
    AND \timestamp\ <= ?`,
  [src, shiftedCutoff]
);

if (rows.length > 0) {
  console.log([AutoClose] Found ${rows.length} candidates for ${src});
}

for (const r of rows) {
  await conn.beginTransaction();

  try {
    // 3. CAPTURE SHIFTED TIME (-5.5 HOURS) FOR UPDATES
    // We write 06:30 so the DB/Grafana displays 12:00
    const shiftedNow = getShiftedTime(new Date());

    // Update status
    await conn.execute(
      `UPDATE alerts SET status='AUTO_CLOSED', \timestamp\ = ? WHERE alert_id=?`,
```

```
          [shiftedNow, r.alert_id]
        );

        // Log event
        await conn.execute(
          `INSERT INTO alert_events
           (alert_id, event_type, old_status, new_status, timestamp, metadata)
           VALUES (?, 'AUTO_CLOSED', ?, 'AUTO_CLOSED', ?, ?)`,
          [r.alert_id, r.status, shiftedNow, JSON.stringify({ reason: 'timeout', src })]
        );

        await conn.commit();
        console.log([AutoClose] Closed alert: ${r.alert_id});

      } catch (err) {
        await conn.rollback();
        console.error([AutoClose] Failed to close ${r.alert_id}:, err.message);
      }
    }
  } catch (err) {
    console.error("[AutoClose] Critical Error:", err);
  } finally {
    conn.release();
  }
}

// ------------------------------------------------------
```

```
// AUTO DELETE WORKER
// -------------------------------------------------------
async function autoDelete() {
  const conn = await db.getConnection();

  try {
    // --- THE FIX ---
    // 1. Get Real Cutoff: Now (12:00) - 5 mins = 11:55
    const realCutoff = new Date(Date.now() - (DELETE_AFTER_MINUTES * 60 * 1000));

    // 2. Shift it (-5.5h): 11:55 -> 06:25
    // We must compare against the "Shifted" time stored in the DB
    const shiftedCutoff = getShiftedTime(realCutoff);

    const [rows] = await conn.execute(
      `SELECT alert_id
       FROM alerts
       WHERE status = 'AUTO_CLOSED'
         AND \timestamp\ <= ?`,
      [shiftedCutoff]
    );

    if (rows.length > 0) {
      console.log([AutoDelete] Found ${rows.length} candidates to delete);
    }

    for (const r of rows) {
      await conn.beginTransaction();
```

```
      try {
        await conn.execute(DELETE FROM alert_events WHERE alert_id = ?, [r.alert_id]);
        await conn.execute(DELETE FROM alerts WHERE alert_id = ?, [r.alert_id]);

        await conn.commit();
        console.log([AutoDelete] Deleted alert: ${r.alert_id});

      } catch (err) {
        await conn.rollback();
        console.error([AutoDelete] Failed to delete ${r.alert_id}:, err.message);
      }
    }
  } catch (err) {
    console.error("[AutoDelete] Critical Error:", err);
  } finally {
    conn.release();
  }
}
(async () => {
  console.log("Worker started with Synchronized Time Shifts...");
  setInterval(async () => {
    try {
      await autoClose();
      await autoDelete();
    } catch (e) { console.error("Loop Iteration Failed", e); }
  }, POLL_MS);
})();
```

Cost Estimation — Time & Space Complexity

Your system's performance depends heavily on how fast rules are evaluated and how efficiently MySQL queries run. Rule evaluation is effectively O(n) where n is the number of events in the rule's lookback window (e.g., last 30 minutes of overspeed alerts). While this is relatively efficient, database queries usually dominate latency. MySQL should be indexed on fields like alert_id, source_type, status, and timestamp to avoid full table scans. To estimate space requirements, calculate the average size of an alert row multiplied by the expected alerts per day, then multiply by retention duration. Logging through Loki and metrics from Prometheus help you measure write rates and refine storage predictions. When data volumes grow, partition tables by time windows (daily/weekly partitions) and archive old data. This ensures MySQL keeps fast lookup performance even at large scale.

Handling System Failures (Fault Tolerance & Recovery)

Kafka provides reliable buffering between your Alert Processor and downstream consumers, ensuring alerts are not lost even if consumers crash. MySQL stores persistent canonical alert records, so it must run on durable storage and include regular automated backups (logical using mysqldump or full physical backup). Docker Compose should have health checks and restart policies so containers restart automatically on failure; in Kubernetes, readiness and liveness probes handle this. Idempotent producers and consumers prevent duplicate processing when retries occur. You should implement exponential backoff for retrying transient failures and push failing Kafka messages to a dead-letter topic for later inspection. Your architecture should clearly document RTO/RPO targets and provide a step-by-step recovery guide for operators.

Object-Oriented Structure / Language Choice

Your current Node.js implementation uses primarily functional modules. To bring more structure and enforce encapsulation, you can migrate critical logic — such as rule evaluation, DB connectivity, and Kafka publishing — to TypeScript classes. This enables clearly defined interfaces, dependency injection, and better testability while staying within the JavaScript ecosystem. For teams requiring strict OOP principles and enterprise-style reliability, a full rewrite in Java/Spring Boot would enforce strong typing, design patterns, and mature concurrency management — but at the cost of increased infrastructure and development

complexity. TypeScript is the practical middle ground for maintaining speed while gaining structure.

Trade-offs in Your System Design

Your design prioritizes development velocity and clear separation of concerns over strict consistency. Storing the alert first in MySQL guarantees a primary source of truth, while publishing a follow-up event to Kafka ensures downstream services (dashboards, workers, consumers) remain decoupled. This introduces eventual consistency, meaning consumers may see updates slightly after MySQL, but the architecture gains scalability and fault tolerance. MySQL simplifies structured queries and historical reports, while Kafka increases complexity but provides resilience and real-time streaming for analytics. Similarly, keeping the rule engine synchronous inside the processor maintains simplicity and predictability, while moving rule evaluation into Kafka stream processors (like Kafka Streams or Flink) would scale better at high volume but adds operational complexity.

System Monitoring

Observability is already built around the Prometheus–Grafana–Loki stack. Ensure each service exports detailed metrics: request latency, rule evaluation durations, DB query times, Kafka producer errors, consumer lag, and worker loop cycle counts. Use structured JSON logging in Loki, always tagging logs with identifiers such as alert_id or driver_id so developers can trace an alert across services easily. Prometheus alerting rules should fire when error rates spike, Kafka consumer lag grows, DB connection pools saturate, or services restart frequently. These alerts should notify engineers early, before the system degrades. Dashboards should visualize both time-series trends and real-time operational health.

Caching (clarified: Redis is for caching — Kafka is NOT a cache)

Right now, you are using Kafka, and it already gives you some “cache-like support” because it stores recent alert logs and lets your consumers replay messages without hitting MySQL every time. This works fine for your current load because Kafka acts as a temporary buffer and helps reduce pressure on the database. However, Kafka is still a log system, not a real cache — it cannot serve fast key lookups, in-memory aggregates, or sub-millisecond reads.

For a heavy system, especially when dashboards or APIs need fast repeated reads (alert counts, driver summaries, latest event state, etc.), you should add Redis. Redis will sit alongside Kafka and provide true in-memory caching, with O(1) lookups and TTL-based freshness. In this model, Kafka continues supporting your pipeline, but Redis handles the heavy read load and prevents MySQL from becoming a bottleneck.

Error & Exception Handling

Each Express service should use a centralized error-handling middleware to catch and format errors uniformly. Wrap all async handlers so rejected promises do not crash the process. Error logs should be structured JSON with type, message, stack, and identifiers like alert_id for easy tracing. Expose error counters via Prometheus so spikes in failures can trigger alerts. Transient MySQL or Kafka issues should be wrapped in retry helpers with exponential backoff to avoid hammering the services. Implement a circuit breaker pattern for downstream systems so a failing dependency doesn’t cause a cascade failure. Maintain a documented list of known failure modes, common error messages, and steps to resolve them quickly — this is essential for operational readiness.