Technical Architecture: Centralized Event Review Platform

****Version:**** 1.1

****Date:**** May 6, 2025

****Status:**** Proposed

1. Introduction

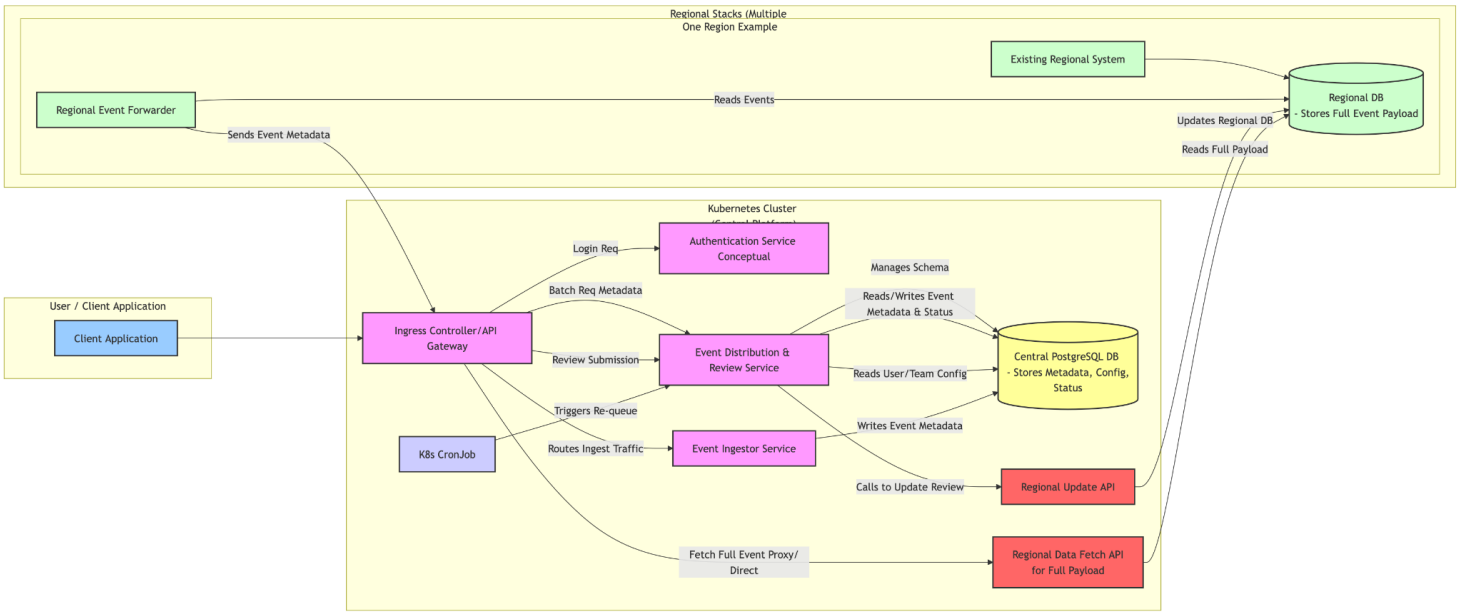
This document outlines the technical architecture for the Centralized Event Review Platform. The platform aims to provide internal engineering teams with a unified interface to review and rate events generated by edge devices deployed across multiple regional AWS stacks (e.g., US, CA, APAC, EU). A key design consideration is to meet regional data compliance requirements by ensuring sensitive regional data remains within its originating region, while still enabling a centralized review workflow.

2. Core Architectural Principles

- **Microservice-Oriented:** The central platform is composed of distinct, independently deployable microservices to promote scalability, maintainability, and technology flexibility.
- **Centralized Workflow, Decentralized Data:** Core review workflow management (assignment, status tracking) is centralized. However, the full, potentially sensitive, event payloads remain in their originating regional datastores. The central platform primarily handles metadata.
- **API-Driven:** Communication between services, regional forwarders, and the client application is managed via well-defined APIs.
- **Containerization & Orchestration:** Services are designed to be containerized (Docker) and orchestrated by Kubernetes for scalability, resilience, and consistent deployments across environments.
- **Data Compliance by Design:** The architecture prioritizes keeping sensitive regional data within its designated region, with the central platform accessing or referencing it as needed without persistent central storage of the full payload.

3. High-Level Architecture Diagram

The following diagram illustrates the major components and their interactions:



4. Component Descriptions

4.1. Central Platform Components

- **Client Application:** The frontend through which users log in, view assigned event metadata, request full event details (via the platform), and submit their reviews (ratings, comments, approval/rejection).
- **Ingress Controller / API Gateway (e.g., Nginx Ingress):**
  - Single entry point for all external traffic (from clients and regional forwarders).
  - Handles SSL termination, routing requests to appropriate backend services based on path/host.
  - Can enforce policies like rate limiting, authentication at the edge.
- **Authentication Service (Conceptual):**
  - Responsible for user authentication and session management (e.g., issuing JWTs).
  - (For POC, this is stubbed/bypassed, with user identity passed via headers).

- **Event Ingestor Service:**
    - Provides an API endpoint (e.g., /api/v1/internal-ingest/event) for Regional Event Forwarders.
    - Receives event *metadata* from various regions.
    - Validates incoming metadata.
    - Stores the metadata in the Central Data Store with an initial 'Pending' status and region of origin.
- **Event Distribution & Review Service:**
    - The core workflow engine.
    - Handles user requests for event batches (metadata).
    - Assigns unique batches to users based on team/region configurations, ensuring no concurrent assignments using database-level locking.
    - Processes review submissions:
        - Calls the appropriate Regional Update API to record the review outcome in the regional datastore.
        - Updates the event's status in the Central Data Store to 'Completed' (or manages deletion for stricter compliance).
    - Exposes an internal endpoint (e.g., /api/v1/internal/trigger-requeue) for the CronJob.
    - Manages the database schema for the Central Data Store via migrations (node-pg-migrate).
- **Central Data Store (PostgreSQL):**
    - Stores event *metadata* (not full payloads), including event_id (central UUID), external_event_id, region_code, status ('Pending', 'Assigned', 'Completed'), assignment details (assigned_user_id, assigned_at), and review outcomes (review_user_id, reviewed_at, review_decision, review_comment).
    - Stores user profiles, team configurations, and team-to-region mappings.
- **Kubernetes CronJob:**
    - Periodically calls an internal endpoint on the Event Distribution & Review Service to trigger the re-queuing logic for events that have timed out or whose assigned user session has expired.

4.2. Regional Stack Components

- **Existing Regional System:** The source system generating events.
- **Regional Database:** Stores the full, original event payloads, adhering to regional data residency and compliance requirements.
- **Regional Event Forwarder:**
    - A lightweight component/agent deployed in each region.
    - Monitors the Regional Database or event pipeline for new events designated for review.
    - Extracts only the necessary *metadata* (non-sensitive summary or reference).
    - Securely sends this metadata to the central Event Ingestor Service via the central

Ingress.

- **Regional Update API:**
  - An API endpoint exposed by each regional stack.
  - Called by the central Event Distribution & Review Service to write back review outcomes (approval/rejection, ratings, comments) to the original event record in the Regional Database.
- **Regional Data Fetch API:**
  - An API endpoint exposed by each regional stack.
  - Called by the Client Application (likely proxied through the central Ingress) to retrieve the *full event payload* for display to the user during the review process, using metadata like external_event_id and region_code.

5. Data Flow Summary

1. **Ingestion:** Regional Forwarder sends event METADATA via Ingress to the central Ingressor Service.
2. **Storage (Central):** Ingestor Service stores this METADATA in the Central DB with 'Pending' status.
3. **Assignment Request:** Client App requests a batch of events (METADATA) from the Distribution Service (via Ingress).
4. **Assignment Logic:** Distribution Service queries Central DB for eligible 'Pending' metadata, assigns a batch to the user, and updates status to 'Assigned' in Central DB.
5. **Full Data Fetch (Review Time):** For review, the Client App (or Central Platform as proxy via Ingress) requests the FULL PAYLOAD from the appropriate Regional Data Fetch API using metadata (e.g., external_event_id, region_code).
6. **Review Submission:** Client App submits the review outcome to the Distribution Service (via Ingress).
7. **Write-Back (Regional):** Distribution Service calls the relevant Regional Update API to store the review outcome in the Regional DB.
8. **Finalize (Central):** Distribution Service updates the METADATA status in Central DB (e.g., to 'Completed' or deletes the metadata record for stricter compliance).
9. **Re-queue:** The K8s CronJob periodically triggers the Distribution Service to check for timed-out 'Assigned' events and revert their status to 'Pending' in the Central DB.

6. Data Compliance Strategy

- **Regional Data Residency:** Full, sensitive event payloads are stored and remain within their originating regional datastores.
- **Central** Metadata **Store:** The central platform primarily stores and processes non-sensitive metadata or references necessary for the review workflow. The event_payload field in the central events table is intended for this metadata, not the full regional payload.

- **On-Demand Full Payload Fetch:** Full event details for review are fetched directly from the regional source when needed by the user, minimizing central storage of sensitive data.
- **Secure Regional Write-Back:** Review outcomes are securely transmitted back to the originating regional datastore.
- **Central Record Management Post-Review:** After successful regional write-back, central metadata records are marked 'Completed'. For enhanced compliance, these records can be configured for archival and eventual purging, or even immediate deletion, based on retention policies and audit requirements.

7. Key Technology Choices (POC)

- **Backend Services:** Node.js with Express.js and TypeScript.
- **Central Database:** PostgreSQL.
- **Database Migrations:** node-pg-migrate.
- **Containerization:** Docker.
- **Orchestration:** Kubernetes (Minikube for local POC).
- **Ingress Controller:** Nginx Ingress Controller (Conceptual).

8. Scalability & Maintainability

- **Microservices:** Allow independent scaling and development of components.
- **Kubernetes:** Provides auto-scaling, self-healing, and rolling updates.
- **Stateless Services:** Application services are designed to be stateless where possible, relying on the Central DB for state.
- **Database:** PostgreSQL can be scaled (e.g., read replicas, connection pooling) as needed.
- **Asynchronous Processing (Future):** For very high ingestion rates, message queues could be introduced between forwarders and the ingestor, or for other background tasks.

9. Future Considerations

- **Full Authentication & Authorization:** Implement robust OAuth2/OIDC-based authentication and fine-grained role-based access control (RBAC).
- **Enhanced Monitoring & Alerting:** Integrate comprehensive logging, metrics (Prometheus, Grafana), and tracing (OpenTelemetry, Jaeger/Zipkin).
- **Client Application:** Develop the full-featured client application for reviewers.
- **Advanced Regional API Integration:** Standardize and secure regional APIs further.
- **Data Archival/Purging Strategy:** Formalize policies for central metadata retention.
- **Configuration Management:** For production, use more sophisticated configuration management tools