🚀 Architectural Design Challenge: Cold Member Activation Feature

Challenge Goal

Design the **Backend (BE) architecture** for a new engagement feature within the Qantas Pay App. The system must reliably track member activity to achieve a core business objective: converting a significant cohort of "cold" members (enrolled but unused white cards) into active users.

The final deliverable is an Architectural Design Document and Diagram detailing the services, data flow, and technology choices required to meet the high availability and financial data integrity demands of this system.

🎯 Business Objectives and Functional Requirements

The feature focuses on a targeted in-app experience with a single, clear incentive:

Objective	**Requirement**
Goal	Load a cumulative amount of **$1000 AUD** (or AUD equivalent foreign currency) onto the Qantas Pay card **within 30 days of enrollment**.
Reward	**2000 bonus Qantas Points** upon successful completion.
Eligibility	Members must load funds within the first 30 days of joining. The loaded funds **must not be withdrawn** (to another cardholder or external account) during that 30-day period.

Key Constraints & Data Sources

1. **Load Types:** The system must handle **Instant Load** (e.g., Debit Card, Apple/Google Pay) and **Delayed Settlement** (e.g., BPay, Bank Transfers).
2. **Backend Grace Period:** The BE must include a built-in grace period to incorporate **late-settling funds**.
3. **Data Warehouse Feed:** Qantas Pay receives all customer transactions **overnight** from its data warehouse, which must be incorporated into the load tracking.
4. **Completion:** The "Congrats!" moment occurs once all funds have been **settled** and the cumulative total is confirmed.
5. **Tracking:** The user must be shown a **confetti moment** and an updated total when the system confirms settlement.

🛠️ Task: Deliverables for Principal Software Engineer

Design an end-to-end cloud-native architecture focusing on the Backend (BE) services. Your solution should include the following sections, ideally presented in a concise document (5-7 pages) and supporting diagrams:

1. Architectural Diagram & Service Definition

- A **clear, high-level system diagram** showing all components, cloud services (AWS, GCP, or Azure), and the flow of both real-time and overnight data.
- Define the core **microservices** and their specific responsibilities.

2. State Management & Data Integrity

- Propose the **state machine** (e.g., Enrolled, Active, Pending Reward, Complete) for a member within the 30-day window.
- Detail how you will handle and reconcile the two disparate data streams (**Instant Load** vs. **Overnight Settlement**) to calculate the single, accurate cumulative total.
- **Justify** your choice of database(s) for the core load tracking data.

3. Time Management & Failure Handling

- Design a **reliable, scalable mechanism** to execute the final eligibility check precisely when the **30-day window expires** for each member.
- Describe how the system handles the **Withdrawal Rule** (anti-fraud check). How is this tracked, and at what point does a withdrawal disqualify a member?
- Outline the strategy for **idempotency and error handling** for the critical final step: the **reward issuance** (to ensure points are never double-issued or missed).

4. Technology Justification & Trade-offs

- Provide a strong **justification** for your primary technology choices (e.g., why Kafka over SQS, why DynamoDB over PostgreSQL, or vice versa).
- Clearly articulate **one major architectural trade-off** you made (e.g., latency vs. consistency) and explain why it provides the best outcome for the business goal.

💡 Assessment Focus

Your solution will be assessed based on its **scalability, resilience, integrity,** and the clarity of your decision-making. We are particularly interested in your approach to **financial data reconciliation** and **managing reliable time-bound state** in a distributed system.