Review Summary

Strengths:

- Technically accurate and well-organized.
- Demonstrates thought leadership in managing Salesforce platform event issues.
- Outlines a solution in actionable steps with measurable benefits.

Recommendations:

1. **Remove any implied client specificity** – while there's no explicit client name, the tone suggests this might have been written for internal or client-specific documentation.
2. **Tone polishing** – the current draft reads like a tech spec; softening language slightly and improving transitions makes it more engaging for a general audience.
3. **Improve structure clarity** – breaking up dense sections and using active voice improves flow.
4. **Clarify reusable patterns** – frame the solution as best practice, not just a reactive fix.

Refined & Public-Ready Version

Case Study: Preventing Platform Event Overload and Recursion in Salesforce

Overview

Salesforce Platform Events offer powerful, asynchronous communication between internal and external systems. However, they are bound by strict system limits that, when breached, can disrupt core business operations:

- **Hourly Publishing Limit**: Up to 250,000 events/hour.
- **Daily Delivery Limit**: Varies by Salesforce edition.

This case study outlines a generalized strategy for addressing issues related to recursive triggers and excessive event generation—common causes of platform event limit breaches.

The Challenge

In complex Salesforce implementations, event-driven architectures can inadvertently create loops that lead to platform instability. Two recurring issues often surface:

1. Cyclic Platform Event Triggering

Platform events trigger other events in a loop, unintentionally multiplying event volume. This can cause:

- Limit breaches (hourly or daily).
- Slower system performance.
- Failure of downstream processes dependent on event delivery.

2. Record-Triggered Flow Recursion

A record-triggered flow publishes a platform event that performs a DML operation on the same record. This re-triggers the flow—resulting in:

- Infinite execution loops.

- Rapid consumption of event publishing limits.

Solution Strategy

To mitigate these problems, a structured approach was implemented:

1. Smart Event Handling Logic

- Applied conditions to ensure events are only published when meaningful changes occur.

- Introduced flags or custom fields to track record state (e.g., “Processed” or “Handled” booleans).

2. Preventing Recursive Flow Calls

- Embedded decision elements in flows to check record status before performing DML.

- Used conditional logic to avoid updates on already-processed records.

3. Flow Design Optimization

- Refactored flows to remove redundant triggers.

- Shifted non-critical updates to scheduled or batch flows to reduce real-time dependency.

4. Custom Rate-Limiting Controls

- Implemented debounce logic to delay or limit rapid, repeated triggers.

- Designed logic to throttle platform event publication during high-volume periods.

Outcome

This strategic refactoring led to:

- **Elimination of Recursion Loops** through decision checks and event guards.
- **Reduced Event Volume** by eliminating unnecessary triggers.
- **Stabilized Performance** during peak usage hours.
- **Protected Downstream Processes** that rely on timely event delivery.

Key Benefits

- **Efficient Use of Platform Events**: Preventing loops ensured sustainable scalability.
- **Enhanced System Performance**: System load was minimized by eliminating redundant event processing.
- **Greater Flexibility**: Custom logic allowed for quick adjustments without deployments.
- **Future-Proofing**: Monitoring and alerting mechanisms provided early warnings for event threshold breaches.
- **Operational Continuity**: Business-critical processes remained uninterrupted, even during spikes in activity.