PRODUCTION RESCUE CASE

Stabilizing a ~10M-Connection Critical Delivery System

A production incident where containment, failure boundaries and business continuity mattered more than finding the bug first.

SCALE	BUSINESS PATH
~10M concurrent long-lived connections	**Real-time order delivery to couriers**

CONTEXT

The platform maintained approximately 10 million concurrent long-lived client connections and sat on a business-critical path. Large-scale connection failures could affect whether couriers received orders in real time.

INCIDENT

During peak load, memory usage on part of the fleet began climbing continuously. The failure pattern was consistent with a hidden connection or resource leak. Waiting for a complete root-cause analysis risked allowing the issue to grow into a much larger production incident.

DECISION

Containment first.

The immediate priority was to limit blast radius and preserve service continuity - not to prove the root cause before acting.

- Used controlled rolling restarts to reduce immediate production risk.
- Kept the service available while investigation continued.
- Separated urgent stabilization from permanent remediation.

WHY THIS MATTERED

In a critical production incident, the first question is often not "What is the bug?" It is "How far can this failure spread, and how do we stop it before the business impact becomes unacceptable?"

ARCHITECTURE LESSON

Production architecture is not mainly about diagrams or technology choices. It is about failure boundaries, recovery behavior, observability, capacity and making correct decisions when the system is already under pressure.

FROM INCIDENT EXPERIENCE TO ARCHITECTURE REVIEW

What I look for before a system becomes an expensive incident

The same production judgment applies to fast-built AI and SaaS systems moving from prototype velocity to operational reality.

A PRODUCTION READINESS REVIEW ASKS:

Failure boundaries	Where can this system fail badly, and how far will the impact spread?
Recovery behavior	What happens under partial failure, restart, backlog, overload or dependency loss?
Operational visibility	Will the team know early enough that the system is degrading?
Capacity & cost	Which assumptions fail first as usage, data or AI workload grows?
Decision reversibility	Which architecture choices are expensive to undo after production adoption?

TYPICAL OUTPUT

- A prioritized architecture risk map - not a generic list of best practices.
- Clear separation between urgent stabilization, medium-term remediation and changes that should not be made.
- Explicit trade-offs for critical technical decisions, including operational and business consequences.
- An implementation sequence the team can execute without pausing product development for a rewrite.

WHEN TO CALL ME

Your team is shipping faster than the architecture is maturing - and reliability, scalability, technical debt or production risk is starting to become expensive.

Cain Wang | Independent Principal Architect | cainarch.com