Tell me about a time when you suggested or implemented a change with the customer in mind/What is an example of a time when you changed direction because of customer feedback?

Situation
We were developing an event stream to publish database changes via SNS using Protocol Buffers (Protobuf), which was ideal for performance. However, I realized that our largest stakeholders might prefer a more widely supported format—particularly JSON—before we committed to this approach. Although pausing to involve them could affect our onboarding timeline, I decided this customer-centric investigation was essential for long-term success.

Task
I took ownership of gathering requirements from our key clients. I organized meetings and demos to validate how well a Protobuf-based solution would integrate with their environments. Through these discussions, it became clear their infrastructures were heavily optimized for JSON, highlighting a mismatch between their needs and our initial approach.

Action

1. **Stakeholder Alignment**: I convened a cross-functional session, bringing together our internal engineering team and the primary external stakeholders. I facilitated an open conversation about the trade-offs between continuing with pure Protobuf and introducing a JSON-compatible pipeline.
2. **Proposed Solution**: I led the design of an additional microservice that would handle the translation from Protobuf to JSON. This preserved the performance benefits of Protobuf within our system while delivering JSON to the customers' services.
3. **Design Review & Buy-In**: I hosted a formal review to present the new architecture. By clearly outlining changes in latency, scalability, and implementation details, I secured team-wide and stakeholder approval for the pivot.

Result
The final solution offered the best of both worlds: rapid internal communication via Protobuf and seamless downstream consumption via JSON. This greatly reduced the onboarding burden for our customers, who were able to integrate the event data with minimal effort. By leading the design and ensuring stakeholder needs were central to our process, I demonstrated both customer-focused leadership and adaptability under tight timelines.

Can you give me an example of a tough technical challenge you've had?

Collaboration & Communication + leadership

Situation:
We experienced a bug where duplicate workflows were triggered on a sheet when using our new webhooks service. The bug surfaced after we migrated from our old webhooks—which provided an *exactly once* guarantee—to the new webhooks, which provide an *at least once* guarantee. The duplication led customers' workflows to run (and thus be billed) twice for each callback.

Task:
I was the lead for the new webhooks migration and the primary point of contact for debugging. The primary goal was to determine why the internal client was receiving duplicate events, given that our

logs showed only a single webhook delivery. Swift resolution was critical because our customers were being double-billed.

Action:

1. **Client Consultation**: I scheduled a call with the internal client to get a detailed overview of their workflow logic, how they detected duplications, and to verify when the two events were triggered.
2. **Code & Logs Investigation**: Although our webhooks logs indicated only one delivery, the internal client's logs and metrics appeared to show two. I worked with the client to walk through their code and see how they were processing events, focusing on how they handled duplicates.
3. **Datadog Analysis**: I examined the client's Datadog dashboards for a historical record of incoming messages and noticed discrepancies in message counts. This prompted deeper investigation into their newly migrated system.
4. **Discovery of Configuration Issue**: The client mentioned they had migrated to a new system but had supposedly rolled back. On closer inspection, we discovered the rollback had not completed properly—a misconfiguration in their scripts was still allowing some new system logic to trigger a second event.

Result:
Once the script error was fixed, the duplication problem disappeared, and our new webhooks service worked as intended. Because we acted quickly, only a small number of customers were affected, and the billing issue was promptly resolved. This cleared the path to proceed with our webhooks migration without further complications.

When was last time you were inspired by someone?

Situtation:

This year. I've worked with a senior engineer on my team, I've admired his project management skills and the way he thinks. An example would be that he said its good to treat the companies money like your own money because it leads to optimisations in code. A situation, where this specific line of reasoning helped was that he was reading aws docs and realised that SNS was being throttled and technically we hadn;t reached the advertised limit.

He then opened up a case and requested the limit be reasied and went back and forth and lowered the costs of SNS by getting a refund and improbing speed.

I like his deeply analytical mindset in every small thing he does which leads to an excellent product. Another thing I like is the way he manages completion dates as well which has improved my own estimatgions much better,

How do you handle conflicts

SITUATION:
- Critical project to migrate webhooks system to a more scalable solution
- Conflict between management and engineers on migration order

- Project impacted both internal and external customers
- Aimed to increase capacity from 20,000 rows to 5 million

TASK:
- Resolve conflict between management's desire to migrate traffic first and engineers' concerns about potential issues
- Ensure the migration process minimizes risks and maximizes efficiency
- Consider both short-term impact and long-term implications of the migration strategy

ACTION:
- Documented all potential solutions with their implications and pitfalls
- Organized a company-wide meeting with engineering teams and management
- Consulted largest internal webhook consumers (30% of traffic) for stakeholder input
- Presented findings and facilitated open discussion in the meeting
- Maintained focus on customer impact and long-term company benefits
- Negotiated compromises to meet quarterly goals while addressing concerns

RESULT:
- Reached consensus to migrate APIs first, then persistence layer, and finally traffic
- Executed migration with zero issues, despite slightly longer timeline
- Achieved dramatic performance improvement: latency reduced from 15s to 3s (95th percentile)
- Set new standards for service quality and scalability
- Successfully met and exceeded both internal and external customer expectations

Tell me a time when you made a mistake and how you corrected it?
Situation

I was implementing a new event type closely related to another event type. This caused a issue where there were cases the other event type wasn;t emitted. This lead to a escalation,.

Task
Our immediate goal was to prevent further event loss and restore system stability. We also needed to ensure no data corruption or incorrect billing (since the duplicate tasks could lead to multiple charges).

Action

1. **Immediate Hotfix**: As soon as we detected the issue, I rolled back the deployment and ensured that the system was stable
2. **Root Cause Analysis**: I dove into the logs and code to pinpoint the exact line that caused this . I discovered that the logic for acquiring task locks wasn’t implemented correctly, allowing two threads to “claim” the same task.
3. **Code Correction & Testing**: I refactored the locking mechanism to ensure proper mutual exclusion. I also added comprehensive test cases that simulated high concurrency scenarios, including stress tests.
4. **Collaboration**: Throughout, I kept stakeholders updated on our progress and collaborated with the QA team on additional test coverage to catch similar issues earlier.

Result
Within 24 hours, we deployed a patch that eliminated the issue. We replayed the events for the past

5 minutes so no data loss other than our SLO being breached. Beyond resolving the immediate issue, we integrated stricter concurrency checks into our code review process and improved our testing environment with more robust load and stress simulations.

What I Learned

- **Concurrency is Tricky**: Even seemingly minor oversights can have serious consequences in parallel systems.
- **Importance of Thorough Testing**: Incorporating stress tests that simulate heavy concurrency is essential to catching these issues before production.
- **Communication & Transparency**: Keeping both internal stakeholders and customers informed fosters trust and makes resolving mistakes smoother.

Growth

Situation
I noticed our event streaming system lacked clear signals to distinguish between a quiet source database and a database outage. This uncertainty occasionally resulted in late detection of issues—especially problematic in a real-time environment.

Task
As a senior engineer tasked with enhancing observability, I aimed to implement a robust method to detect and alert on potential outages in near real-time. My plan involved introducing replication heartbeats and user canaries so we could confirm that events were truly flowing—or identify when they were not—across the entire pipeline.

Action

1. **Technical Investigation**: I researched existing libraries and discovered that our Go-based system could leverage the **go-mysql** library to simulate replication behavior. By configuring the event streaming system to act as a replica, we could enable regular heartbeats from the database to the streaming pipeline.
2. **Design & Documentation**: I detailed how these heartbeats, combined with lightweight "canary" events (artificially generated test messages), would allow us to differentiate between a genuinely quiet database and a connectivity or replication issue. I also prepared documentation explaining the architectural changes, monitoring hooks, and alerting logic to ensure cross-team alignment.
3. **Implementation**: I wrote the code to integrate the replication heartbeat mechanism into our existing streaming service, ensuring backward compatibility with current consumers. I also added user canaries as an additional layer of verification.
4. **Deployment & Validation**: After thorough testing in our staging environment, I rolled out the solution to production. I worked closely with our DevOps team to configure dashboards and alerts, so we'd receive immediate notifications if heartbeats or canaries went missing.

Result
The improvements dramatically increased our visibility into the health of the pipeline. We can now swiftly determine whether the database is simply idle or if there's a legitimate outage. This early

detection capability has reduced our mean time to resolution (MTTR), minimized downtime for our internal and external stakeholders, and overall improved trust in our event streaming platform.