Architecture Overview

For this prototype, I approached the problem the same way Cloudflare products are usually built: small, composable pieces that can scale later, even if the first version is simple.

At a high level, I imagined this as a feedback pipeline that starts at the edge, cleans and enriches the data, and then presents it in a way that helps teams quickly understand what customers are experiencing and what needs attention.

I used Cloudflare Workers as the entry point because they're a natural fit for ingesting feedback from lots of different places. Support tickets, GitHub issues, community posts, and social channels all produce data in different formats, and Workers are a clean way to normalize that data close to where it enters the system without managing servers. In this design, Workers would tag each piece of feedback with basic metadata like product area and source before passing it along.

To make the system more reliable, especially during spikes in feedback volume, I placed Cloudflare Queues between ingestion and processing. That way feedback doesn't get lost if downstream processing slows down. It also makes it easier to scale individual steps independently instead of everything being tightly coupled.

For deeper analysis, I layered in Cloudflare's AI capabilities. Keyword matching alone is fast but limited, so AI adds a second pass that can group feedback into more intuitive themes, identify frustration or urgency, and generate short summaries that are easier for humans to scan. In this prototype, those AI calls are represented as a stub to show how the integration would work without requiring real API keys.

Once feedback is normalized and enriched, it needs to live somewhere that's easy to query and slice. I chose Cloudflare D1 as the conceptual storage layer because a relational model works well for this kind of analysis. It makes it straightforward to ask questions like which products are seeing the most customer pain, how issues trend over time, or whether certain themes are getting better or worse.

Because this tool could realistically be used by both internal teams and selected customers, I assumed Cloudflare Zero Trust would sit in front of it in a real deployment. That would allow Cloudflare employees to see the full dataset, while customers could be limited to views that only include their own feedback. The access model stays clean without exposing raw data broadly.

Finally, the dashboard itself is built in Streamlit purely as a prototyping choice. It lets me demonstrate how insights would be surfaced without getting bogged down in frontend complexity. The dashboard uses synthetic data so it looks live and interactive, but the structure is designed so it could easily be wired up to real APIs or a database later.

Overall, the goal wasn't to build something production-ready, but to show how Cloudflare's existing products could work together to turn noisy, scattered feedback into something teams can actually act on.