

Telstra × SnapJQL

Customer story - narrative for review

Spoken by [Name], [Job title], Telstra · Draft, September 2026

1. The environment we work in

I'm [Name], [Job title] at Telstra, and I work with our Jira Cloud environment day to day.

To give you a sense of scale: we're talking about roughly twenty thousand users. At that size, Jira stops being a tool that a few teams use and becomes part of how the organisation actually runs. Work is spread across programmes, across levels of hierarchy, across linked issues. People need to find things and they need to be able to find them themselves, without raising a request and waiting for somebody to build them a report.

That's the context for everything else I'm going to say. None of this was a pilot or a side project. It was a change to something a very large number of people depend on every day.

QUESTIONS

- *Can you describe the Jira Cloud environment your team is responsible for, and roughly what scale it runs at?*
- *What does advanced search actually need to do for people at that size?*

2. What we needed from the next step

We were already using another Jira Cloud plugin for advanced search. It had served us. But when we came to look at what we would use next, our requirements were not the same ones we'd had the first time round.

Two things mattered more than the feature list.

The first was architecture. We needed a Forge-native application, and data residency was part of that conversation from the very beginning - not something raised at the end. In an organisation like ours, where an app runs and where the data sits isn't a detail you tidy up later. It's the gate you have to get through before anyone even looks at what the product does.

The second was the search capability itself. We needed multi-level hierarchy queries, the ability to filter on properties across those levels, and proper

linked-issue functions. And we needed all of that inside native JQL in the issue navigator, in filters, in automation rather than in a separate interface sitting alongside Jira.

QUESTIONS

- *What were your must-haves going into this, and why did architecture come before features?*
- *Why did Forge-native and data residency matter so much for an organisation like Telstra?*

3. Why SnapJQL

SnapJQL met both sets of requirements, which turned out to be a shorter list than you might expect.

It extends Jira's own search rather than replacing it. The hierarchy queries, the cross-level property filtering, the linked-issue functions they behave like JQL, so they work everywhere JQL works. That mattered to us. The moment a capability only exists inside a vendor's own screen, adoption stops at the people who know that screen is there. It also meant we weren't asking teams to learn a new way of working - if you already knew JQL, you already knew most of this.

And it's built on Forge, which answered the architectural question directly rather than by exception.

So it went forward. But I'd be honest that at the point we chose it, we were choosing on architecture and capability. What we didn't know yet was how it would behave at our scale — or what the team behind it would do when it didn't.

QUESTIONS

- *What made SnapJQL the right fit on paper?*
- *At the point of decision, what did you still not know?*

4. Scale was the real test

This is the part I'd most want other people to hear.

SnapJQL had worked well in smaller environments - it had been tested and run at something in the region of one to two thousand users. Ours is roughly ten times that.

An order of magnitude isn't a bigger version of the same problem. It's a different problem. Query patterns that are perfectly well behaved at two thousand users don't necessarily behave the same way at twenty thousand. Edge cases that

never surface in a test environment surface on an ordinary Tuesday morning, because a few hundred people did something nobody had thought to try.

So we went in expecting to find things. And we did — behaviours and edge cases that a one-to-two-thousand-user test environment simply would not have exposed.

QUESTIONS

- How different was your environment from the scale the product had been proven at?
- What kinds of things only showed up once it was running at your scale?

5. What happened when we found problems

This is what actually changed my view of the engagement.

I opened more support tickets than I could ever have imagined - and every one of them got a response.

I want to be precise about what I mean by that, because good support is something every vendor claims. What I mean is this. When we raised something, it wasn't triaged into a category and closed. Snapbytes went into the logs with us. They reproduced the behaviour. Where the cause sat in their product, they changed the product. Where it turned out to sit on the platform side, they documented it, built the evidence and escalated it to Atlassian — rather than telling us it was outside their scope.

That line - inside the app versus outside the app - is where a lot of vendor relationships quietly end. It didn't end there. They stayed with the issue until the root cause was understood, whichever side of the line it fell on.

QUESTIONS

- What was it like raising issues with Snapbytes?
- Was there a moment where you expected to be told "that's not our problem"?

6. Where that leaves us

The SnapJQL we're running today isn't the SnapJQL we started with, and that's rather the point.

The things we surfaced at our scale went back into the product. It got better because it was being used here, under real conditions, by people who weren't being gentle with it. We weren't testing it in a lab — we were the environment that found the problems.

The other thing worth saying is that none of this was dramatic. There was no crisis and no rescue. It was a long run of small, specific problems being taken seriously, one after another, which is honestly what you want from a supplier.

For us the outcome is simple enough to say in one sentence. We have advanced Jira search that works at our scale, on an architecture we were comfortable approving, backed by a team that treats our problems as their engineering work rather than as tickets.

QUESTIONS

- *What's different for your teams now, compared with where you started?*
- *How would you describe the working relationship today?*

7. What I'd tell someone evaluating this

If you're a large organisation looking at this kind of app, I'd say two things.

Ask about architecture early. Forge-native, data residency, where the data actually sits — that should be your first conversation, not your last. It is far cheaper to rule something out at the start than to discover the problem after people have begun depending on it.

And ask what happens when something breaks at your scale, rather than at demo scale. Feature lists are easy to compare side by side. What you can't see from the outside is whether the team behind the product will still be engaged six months in, when you're the one finding the problems and none of them are convenient. That's the part I'd want to know about — and for us, that's the part that turned out well.

QUESTIONS

- *Would you recommend working with Snapbytes, and why?*