SACRA

EXPERT INTERVIEW

UPDATED
02/02/2023

Ravi Parikh, CEO of Airplane, on building an end-to-end internal tools platform

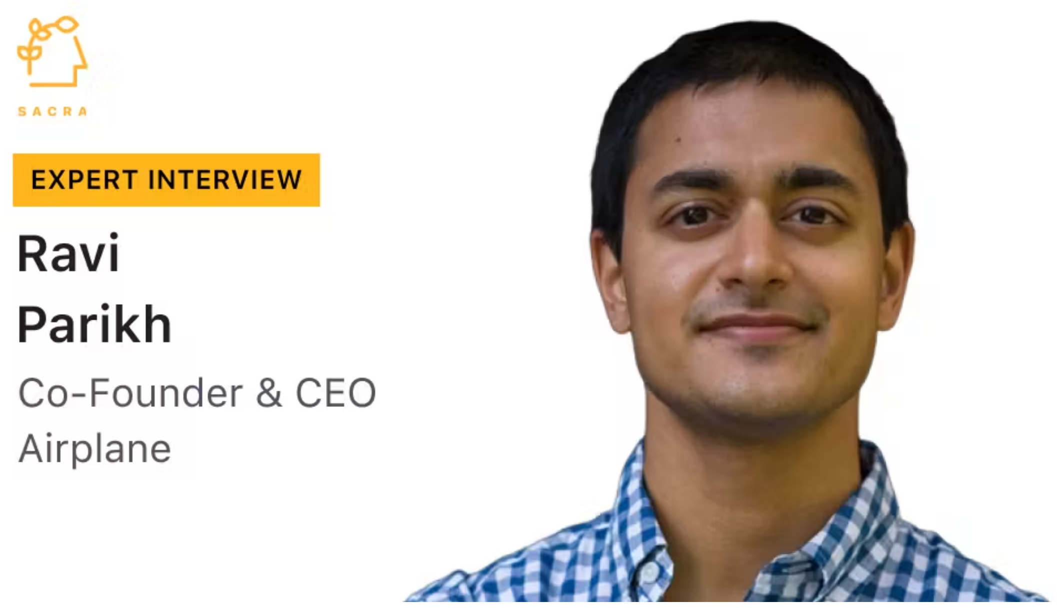
TEAM

Jan-Erik Asplund
Co-Founder
jan@sacra.com

Published on **Feb 02nd, 2023**

Ravi Parikh, CEO of Airplane, on building an end-to-end internal tools platform

By **Walter Chen**



Background

Ravi Parikh is the CEO of Airplane. We talked to Ravi to learn more about the business model of the "PaaS for internal tools" ala Retool and Appsmith and how these players differentiate from each other.

Interview

You previously co-founded the analytics company Heap. Could you tell us what inspired you to start Airplane?

I co-founded Heap in 2013. I'm an engineer by background, so I spent the first couple years of Heap building out the product. Heap is an analytics product—web analytics and mobile analytics. As we scaled up the company, I actually mostly focused on go-to-market. I built our sales team, customer success, marketing, solutions, and engineering teams for most of my time at Heap.

There are lots of things I found that happen in the course of serving a customer that are very complicated to resolve.

For example, a customer might implement the Heap API wrong, and then there's a bunch of messed up data in their account. Someone on our end would have to go in and delete all that data or run some script that fixes things. These kinds of one-off manual operations to serve our customers would come up a lot.

Usually, in such cases, what would end up happening is that they would reach out to our support or success team. Then, someone from there would have to escalate to engineering.

Someone from engineering would have to get involved, run some script, and then supervise that script for the next two hours—because it's very data-intensive—and then wait for that to complete.

This interrupt-heavy pattern of serving customers was increasingly common as we grew.

We had various attempts to try and build better internal tooling. We built this Slack bot that you could use to run some basic commands, but these kinds of systems were always breaking and never were a complete solution. There was also a long tail of one-off scripts that was being used to solve problems. That was the status quo at Heap.

We actually bought Retool at one point, but Retool was mostly not used to solve those problems. We weren’t able to use it for solving these more script-heavy, workflow-heavy, problems because it was just a UI builder.

I left Heap in 2020 when the company was about 200 people, and I spent some time thinking about what I wanted to do next. I was brainstorming ideas with a friend of mine, Josh, who's now my co-founder at Airplane.

Josh’s journey was similar to mine. He was CTO of a company called Benchling, which is a life sciences SaaS company. We realized we had the same set of problems, and Benchling also had found itself dealing with all of these kinds of internal one-off operations that would hit the engineering team.

The initial insight was, “There's a lot of engineering time that goes into being interrupted by other teams to solve one-off issues or building platforms to help teams solve their own issues, i.e. internal tools. The patterns that come up over and over again across all these companies look similar.”

We thought we could build a platform or framework to allow you to build a lot of this stuff a lot more quickly.

The initial version of Airplane was really, really simple. Basically, we’d let you take a script—like a Python script that does some internal operation—and deploy it to our platform. We’d handle putting a UI in front of it, role-based permissions, auditing, and notifications.

These were really basic considerations that were part of why it was hard to take engineering-only operations and share them with non-engineers within Heap and Benchling. That was the initial idea—just script to app. That's what we launched with in mid-2021, and really mostly what the product was for nearly a year after that. Now, we've built out a lot more additional features to help build internal tools, though it took us a while to get there.

Could you talk about what your initial product-market fit looked like? What's been surprising about the core initial users and use cases that you've found?

We got pretty strong initial traction with just that scripts-to-apps idea. We launched it on Hacker News, Product Hunt, and immediately got a decent amount of interest with people signing up for the product. I think the willingness to pay wasn't quite there. We did close a couple larger enterprise contracts here and there, but for the most part, people saw Airplane—as it was then—as one small component of internal tooling that didn't really stand on its own.

Here’s what would happen. Let's say you have a data deletion script and you want to model that in Airplane. You might use Airplane to create a web form that takes in a single “user ID” parameter and kicks off this script. The problem is, you can't really just have that in isolation. The typical support workflow is you're going to want to look up the user first, maybe based on a keyword search or a username. Then, you're going to see a couple rows in the database that match, click on one of the

users, grab their ID, and kick off the script to actually do the deletion. There's this read/diagnose phase before you do a write operation.
What people would end up doing is that they would use Airplane and then they would also use Retool or an in-house tool to have direct access to the database.

They'd copy-paste across tools, so Airplane didn't stand on its own—in the sense that the specific workflow that would happen for a support person or an ops person didn't fully live within Airplane. That was confusing for a lot of end users.

We ended up having to build a way to let users read and explore data as well, which we call Views, and which competes more directly with tools like Retool or Appsmith.

You can use Airplane as an end-to-end internal tooling platform. That's pretty recent, maybe seven, eight months ago that we've really made that available to people. Since then, we have seen a significant uptick in our ability to close larger agreements.

Airplane initially focused on replacing one-off scripts. Why did you go after one-off scripts as the wedge? What did that get you in terms of adoption inside the org?

I think it was the right order of operations. The scripts aspect was really unique. It was something you could not do with any other platform. There's one tool called Rundeck, owned by PagerDuty now, that is similar to what we do. But there's not another thing out there that does the exact same thing as Airplane.

Even Rundeck is really meant for DevOps / SRE type workflows. It's not quite the same. It's a much older product, so it doesn't take advantage of a lot of really modern stuff, like serverless patterns, that make Airplane a lot simpler to set up. That unique hook was really appealing to people.

Even if it wasn't a complete solution, it was a compelling hook. It drove a lot of interest, a lot of signups. For some companies, it did represent a big swath of their internal tooling and strong economic value. We did close a lot of our initial customers for whom this was a burning hot pain point. But we expanded our TAM significantly once we added Views.

Can you talk about how a developer (your customer) thinks about using Airplane vs building a React app vs buying an off-the-shelf SaaS for a particular internal tool?

For the latter things like off-the-shelf SaaS, the vast majority of use cases that people use Airplane for, or at least the ones that come to us and use Airplane, there's usually no off-the-shelf SaaS that would just do it.

No one's building a CRM in Airplane, for the most part. It's mostly bespoke things like, "I need to build this internal admin panel that does these really specific read and write operations against our production database." If you were using Django, maybe you could use Django Admin, or if you're using Rails, you could use Active Admin. But if you're not using one of those specific web frameworks, there's no off-the-shelf SaaS that just does that for you.

In terms of build versus buy essentially, that is the primary thing that people are usually evaluating with Airplane. It's not actually like Airplane versus Retool. Ninety percent of the time it's, "Either we're going to use Airplane for this or we're probably going to build this in-house."

The way we navigate that is, Airplane is pretty different from most of the other products in this internal tooling space, in that we're not a low-code, no-code solution. Airplane is entirely a code-based product.

So, if you're saying, "I really want to have a lot of control over this and I want to use React to build the front-end," you can actually just do that in Airplane. We give you a component library, the state management system, and a bunch of built-ins that make it easy to build internal tools. But you’re still writing React code, at the end of the day, if you’re building a view on Airplane.

We like to think that using Airplane lets you have your cake and eat it, too. You can take advantage of all the niceties that Airplane gives you out of the box in terms of handling permissions and notifications etc. that will accelerate your time to development, while at the same time getting all source control benefits of code. Airplane's fully extensible. You can import your own private libraries or third-party libraries, so you don't have to make that trade-off.

When people look at Airplane versus in-house and still choose to go with in-house, there's usually one of two reasons.

First is, they've already built it. They're like, "Well, we have this legacy system. We know we want to overhaul it. Airplane seems neat, but we don't really want to rebuild from scratch. We'll just incrementally build on this thing we've already done," which is fair and I think that's fine.

Second might be, there's something highly bespoke they want to do which is probably going to be more trouble than it's worth in Airplane. If the UI you want to build internally is really specific, and you're not going to take advantage of our component library at all, then, Airplane's not really giving you that much value. But that's really rare. There's some tables, charts, maps, graphs, some form fields, and buttons and they look the same. That's the Retool thesis as well. But occasionally, someone will have some really specific needs. That’s when I'll be like, "Yeah, maybe just build that in-house. That seems fine."

Airplane leads more with code than a point-and-click UI builder. Why is that? How does your design philosophy help balance power, flexibility vs. off-the-shelf components that save developers time?

We do get a wide spectrum of feedback on the way Airplane works. While obviously this is a biased viewpoint, we actually think Airplane is the fastest way to build these things. We don't think there's a trade-off if you're a developer. The actual trade-off we’ve made is, if you're not a developer, then you can't use Airplane.

Whereas, with Retool, I know that there are the non-developers or the SQL savvy who can drag and drop some things, hook it up with some queries, and get some value out of it. You can't really do that with Airplane. That is a real trade-off we've made. But if you are a developer, even if you're not a React dev, and you know how code works, our belief is that Airplane is faster. We have heard that from people as well.

That being said, not everyone believes that. Some people will look at our docs, they'll glance at the Retool docs, and they'll be, "Intuitively, Retool feels simpler. I'm not a React dev. I'm a backend dev. I don't want to learn React. Airplane seems a

little bit intimidating." To those people, we usually just say, "Go through our 15-minute Getting Started guide and tell me if it feels slower or faster."
Some people will be like, "Look, I'm not a React dev, and I went through the Airplane Getting Started guide, and actually, it felt really easy." That's because a lot of things that make React and front-end web dev hard are things that involve the emergent complexity from building complex web apps. You're not really building all that complex stuff in something like Airplane. All the state management is really simple. As a result of that, we can enable quite a bit of speed.

Our script-based approach also leads to a huge speed boost for developers. Scripts are the fastest ways to solve a problem for a developer. Write a script, deploy it to Airplane and we'll auto generate the UI, the notifications, permissions, the audit logging etc. It actually is significantly faster than having to put that same code into some API endpoint and build a little bit of UI in Retool and have it hit that API. But we do sometimes get that reaction to work where intuitively it feels like writing code will be work. So, we have to thread the needle a little from a messaging perspective.

Then, you also get people saying, "I already know React. This seems really fast." You can get both reactions.

Airplane monetizes on a per seat basis for any user or creator of an Airplane app. Do you have any plans to differentiate between users and creators? Do you have any usage component to pricing in your enterprise tier?

We, like Retool or some of the others in this space, just do naive seat-based pricing. There's no distinction between types of seats.

There's certainly pushback we get where people will say, "If I'm going to roll this out to my whole org, there's going to be teams that use it less than others. There'll be teams that are creating versus those that are just consuming. There are people I just want to view stuff; I don't even want them to run any write-based operations. Why do I pay the same for all of them?"

The way we've solved this for now—it's not a permanent solution—is just that the seats are really cheap. It's $10 for our

Team plan. On our Enterprise plan, I'm not going to say what it is, but it's way lower than all of our competitors.
We're not trying to use cheap as a lever to win deals and like I mentioned earlier, 90% of our deals are build versus buy. It's more, just that at our current stage, we're closing business fast enough that we're not trying to squeeze every dollar out of every customer.

Heap initially had a generous free tier. But once we started scaling sales, we really nerfed the free tier and started monetizing aggressively, and I think that was a big mistake. Amplitude did not do that. Our organic sign-ups at Heap were climbing steadily over time and then, we made that change—they kept climbing for a little bit but then, they leveled off. We just killed that word of mouth effect without knowing that we had killed it at the time. Our revenue did skyrocket, because we started really pricing the product more effectively. But it killed the golden goose a little bit from those free leads we were getting.

It's a deliberate strategy to be like, "Let's leave some consumer surplus on the table, not unnecessarily, but let's just optimize for getting deals done quickly, getting good logos, good case studies, good reference customers, and getting people who are going to love the product and tell their five friends." That's more the strategy right now. We've sidestepped the issue.

In a more medium- to long-term sense, there probably will be some sort of usage-based component to Airplane's pricing partially because the use cases for Airplane are a little different from a Retool or something like that. It's a purely UI-based tool. So, you have like 25 or 100 support agents and that's who you monetize on.

For Airplane, there's a lot of usage of Airplane that isn't really user-based usage. A lot of people use Airplane for scheduled operations. There's not a seat concept that really makes a whole lot of sense if you're running a schedule. But if you're running 100,000 instances of a scheduled job every week, that should be monetized in some sense. Right now, it's not being done. There's actually people who have like five seat accounts that are getting massive amounts of value in compute for free. That is something we have to fix, but that's how we've done it for now.

How do you think about Airplane's positioning vs. Retool? Retool looks more positioned as a SaaS whereas Airplane looks like a PaaS. What are the relative merits and drawbacks of each approach?

So, we have a very self-selected audience of people and our acquisition strategy is all inbound for now. I mean, 10% of the time it is versus Retool but most of the time, they're like, "We've seen Retool before," or, "We've used it in the past," or, "Used it in my last company," and decided not to use it already before ever talking to us.

The reason they don't use Retool sometimes is related to a lack of functionality. We have the scripts and that's unique. But a lot more of the time it's due to the way Retool's built. We'll talk to a lot of people who will be like, "Hey! I'm not really into all the low-code, no-code stuff. We were going to build this internal tool in-house, that was going to be the preferred way to do it. Then, I came across Airplane and realized, this actually might be a best-of-both-worlds sort of thing." So, we get a lot of people who really resonate with Airplane's philosophy.

There's a customer of ours that we closed a month ago where a couple months before that he had tweeted something like, "I want Retool, but actually built for engineers." We reached out to the person and he was like, "Oh! This actually is the thing I was looking for." Within a couple weeks, we closed a large agreement with that customer.

Customers usually know that Retool and many other things exist but they've chosen not to use it, due to being a poor technical fit for the way they need their platform to work. But by virtue of being code-based, Airplane is a fit for them.

Retool is a drag-and-drop system where the underlying representation is a domain-specific language that's very Retool-specific. It's not something you can easily debug, version control, do code reviews against, extend with your own code, and so on. Whereas Airplane is normal JavaScript or Python. You own it. You version-control it. You can stick an Airplane folder in your monorepo and do stuff with it.

That's the big differentiator. By the time they're even talking to us, most people already know that because they've seen our

marketing material and found us via word of mouth. That's the positioning I guess.
From a positioning standpoint, Appsmith open sourced its platform and sells a hosted version, whereas Airplane builds on open source like React but within a proprietary platform. To the extent that Airplane is something of a Django admin for React, have you considered open sourcing Airplane? How do you think about open source vs proprietary as you build Airplane?

While we get this kind of question from people in your position —analysts and investors—customers don't see it that way. The reason is that I think the way Appsmith approached open-source is really strange.

I'm not trying to be insulting here, but I think of it as a weird product. It's open source, and I'm a big fan of open source software, but it's an open source version of essentially Retool, which is a proprietary system. So, an open source proprietary system is, I guess, slightly better than a purely proprietary system.

However, if you build an app in Appsmith, you are using Appsmith's constructs to build that app. The fact that it's open source doesn't actually make it any more portable.

What are you going to do? Clone Appsmith and somehow weave that into your own codebase? You're still building in a low-code silo. The fact that you can inspect the code of the low-code silo doesn't actually help you all that much.

With Airplane though, when you build something in it, it's your own code. It's normal JavaScript, Python, whatever code that you can version-control, store in your own code base, and do whatever you want with it. If you had open source Figma, that doesn't make Figma any more accessible to a developer. It's still an application that you drag and drop things in. It's not like now you have a code representation of your Figma file that somehow that changes things for you.

I think it's halfway there. Open source is good, but they open source the platform, not open source the actual DSL in which things are built in Appsmith. I could be mischaracterizing it. I have not played with Appsmith nearly as much as I've played

with Retool and some of the other tools. But that's my understanding of it.
So, when you take that guy on Twitter who mentioned, "I want something like Retool but for developers," I don't know if he'd seen Appsmith or not. But I think he would look at something like Appsmith and say the fact that it's open source doesn't automatically make it actually a developer tool at its core.

I think the open source strategy has been effective marketing for Appsmith. When I do speak with Appsmith customers, the thing that resonates with them is less that it's open source and more that it's free, by virtue of being open source. So, I think, the open source is a distribution strategy that's been effective, but I don't think it's necessarily been something that has inherent value on its own.

I'm a fan of open sourcing things. We haven't open sourced any components of Airplane. We have debated open sourcing our View component library, since you could actually take Airplane View components and reuse them in your own codebase in a way that’s totally independent of Airplane. It takes some work to actually package it in the right way, but we'll probably do that at some point.

Retool has benefited from being used by large ops teams and distributed gig workers which offers seat expansion. Being indexed on high seat type use cases, is there a parallel for Airplane? Is that a headwind or cause that played out for you?

Probably about half of our users or customers are using Airplane in a way that is very similar to Retool. It's fintech companies with ops teams, companies where they have a bunch of agents—support agents, ops agents, compliance teams—all of whom need to have access to some internal UI, and scripts to run things.

The other half are the things that actually I don't know that people really use Retool for, which is more like eng-only use cases, where it's an engineering team with a ton of on-call runbooks, scripts, and stuff like that, and they want to turn those into a set of operations that are shareable.

A very common thing would be, "Our eng team is growing. We want to shut down this idea that everybody has access to prod,

and we want to just have a set of constrained operations that they can run in Airplane one-off." Those are not nearly as high-seat as this, "We have a 500-person ops team," or something like that but they still can be pretty high-seat. There are 100 person, 500 person, 1,000 person eng teams out there, and they still need things like this.
We're probably undercharging for the seats when we have those eng-only use cases. It is hard to find a one size fits all seat price, but that's also fine. It's what I mentioned earlier, we're not trying to squeeze every dollar out of every contract. At some point, I think those eng use cases tend to incur higher compute and there's other ways we can monetize that later, but for now, that's not really the focus.

Are you envisioning Airplane as a tool with wall-to-wall adoption company, where everybody has a seat and Airplane is a tool that everyone in a company uses? Or, is it more zooming in and charging a higher per seat for engineering use cases? How are you thinking about that on a longer time scale?

More of the former than the latter.

In our top accounts, of which we have quite a few that now fit this pattern, we get basically wall-to-wall adoption. These are tech companies. Other types of companies might be a little bit different. But in a typical tech company, we get close to wall-to-wall adoption from every team except maybe G&A or marketing.

Then, everyone on the eng team has an account, both because they need to build stuff in Airplane, but also because they have these DevOps-y type things, these on-call runbooks, and scripts, they need to run. People on the product team have to do feature flagging stuff in Airplane. So, it's pretty wide. We'll have 200 person companies with like 140 seats; that kind of thing. Our best accounts fall into that pattern.

As Airplane gets more fully featured, it starts to look like a PaaS but for internal tools. What stops a PaaS for internal tools from becoming a generalized PaaS for launching any app internal or external? If Airplane succeeds, will it eventually collide with Vercel and Netlify?

I see the type of platform needed for internal tooling as very distinct from what's needed for customer-facing applications for many reasons. It may not be realistic for us to branch from one to the other anytime soon. Occasionally someone might need to embed an Airplane thing publicly, but it's not going to be the primary use case for Airplane for a very long time, possibly forever.

That's a little different from the vision that companies like Retool express. But fundamentally, the needs for both are really, really, really different. We would have no illusions that we'll ever compete with Vercel or Netlify.

That’s because internal tools have a lot of properties that make them distinct from external apps. The reason that a “framework for building internal tools” is even a sensible concept, while there's no similar framework for building customer-facing apps—there's rails and stuff like that, but these are much lower level.

So, internal tools have a lot of things in common.

One, they have this captive audience effect. If I build a support dashboard, my support team has to use it. They can't be like, "Nah. I don't like it." Maybe they can, I don't know but you don't have to have pixel perfect UI, UX. That's the reason you can have these one-size-fits-all component libraries in Airplane. You don't have to think super deeply about customizing look and feel to a strong degree.

Usually the scale is not super high. What's the most users you could have of an internal tool? Maybe 100,000 for the biggest companies in the world? That’s smaller than even a modestly-sized consumer app.

There’s also this strong focus on permissioning and auth. The whole auth model and the permission model for an internal tool is just totally different than it would be for an external tool. So, everything is built around that.

You have way too many decisions that Airplane has made that are very specific to internal tooling that will be hard to break out of. Now, occasionally someone will be like, "Hey! We're thinking of Airplane as 98% internal tooling, but we also want to build this one portal with Airplane that we expose to some of our enterprise customers so they can see some data." Maybe we'll support something like that. But we're not going to support the ability to build an actual production grade customer-facing app. It’s not realistic for us in the near future.

Can you talk about the sequencing of Tasks, Views, Workflows and what's coming next in terms of the primitives that you're building which enables you to eat up different SaaS markets?

Tasks basically represent the ability to turn compute-based, write-based, script operations into reusable apps that you can run.

Views represent the ability to create UIs really quickly.

Workflows is an extension of tasks. It lets tasks call other tasks and do orchestration and multi-step stuff within a task.

With all that combined, you can read data, write data, compute things. The only thing missing from full-stack app development is storage really.

So, you can build a fully functional app in Airplane as long as you bring the database—whether it's a hookup to Salesforce, Postgres. We're not going to handle that for you. At some point, maybe we will. I know Retool recently released a Retool database or something like that, where it's hosted Postgres.

To be honest, it's not a common customer request. It does come up. There are use cases for which having that natively on Airplane would make our customers' lives easier. But nine times out of 10, people are just hooking up Airplane to an already existing production database or a set of API endpoints on top of that database. For that reason, the set of primitives that we have in Airplane today actually services 99% of what customers ask us for.

I don't know the sequencing of when we'll build more primitives, necessarily. A lot of what we're doing now is more features and functionality on top of what exists in the Airplane —developer experience improvements and that kind of stuff. But it's less fundamental building blocks and more incremental value generation, at least for the foreseeable future.

You mentioned people hook up Airplane to a production database. Have you seen more use cases involve moving around some of these Zapier-esque or ETL-reverse ETL type use cases that can be accomplished with Airplane of moving data between SaaS products, or moving data between SaaS products and production data warehouse etc.?

Very occasionally, people will do stuff like that with Airplane. But Airplane is a platform for basically turning code you've written into reusable apps that people can do stuff with. You can definitely write code in Airplane that hits the Salesforce API, and then takes some data and shoves it into Snowflake.

But there are tools, to your point, that are specific SaaS already, like Zapier or these reverse ETL companies, like HighTouch or Fivetran. They already put data from A to B, and instead of writing code for it, just click three buttons in a UI and it'll do it for you.

Those companies are better at that than Airplane. Airplane is more generic. We have seen people use Airplane for data pipelines but they're usually highly bespoke data pipelines.

Given that you're building a product that writes to production, and that's one of the key aspects of what Airplane does, how do you think about the security side of it?

The level of security scrutiny that Airplane gets is next level compared to anything we had to deal with at Heap just because Heap just tracks client side data. The worst thing Heap can do is, I guess, mess up your website or something so you just uninstall it and it'd be fine.

Airplane has write access to production resources. There's no way around that. That is definitely not a thing that you want to trust lightly to a third-party. Retool does this as well. As a result, Retool offers a fully self-hosted version.

We have a hybrid cloud approach, where you can self-host an agent on your own cloud that does the actual computation but the actual SaaS plane is still in Airplane's cloud. That kind of hybrid approach seems to work well for people. It gets the best of both worlds. You get SaaS level innovation speeds and a lack of maintenance, but there's this small thing that you host yourself and that gives you certain security guarantees about where the data's going to be processed. That helps a lot. That allows people to use Airplane without having to expose databases or API endpoints to the public internet and mitigates a lot of the concerns. At the end of the day, we had to go through SOC II compliance a lot earlier than we ever had to at Heap. So, it's something that people do scrutinize. It's not been a major blocker, but it's just meant that we've had to take certain measures as a company sooner in the lifecycle than a typical SaaS company would.

Airplane is code-based, yet it allows the code to run through some quality control process and have certain guarantees vs. something that's a little more SaaS-y where you could write some text and it would generate some UI. With the UI, you couldn't reason about how it worked or whether it worked.

The debuggability of Airplane is a very strong point and this will be hypercharged in the AI world. But even now, people will come to us and say one of the reasons they like Airplane is because, if they're building a view in Airplane, it's just one file with a bunch of React code in it that they've seen a thousand times before in their life. Whereas, when they're in something like a Retool, they're like, "Well, I have a bunch of parts of the UI that poke out and have some JavaScript code in them. I have to reason about how the data flows between these four components, and that's quite hard to do in an unfamiliar territory."

To your point, if you get a world where I can write some natural language text and it spits out this finished product or a 90% finished product for me, that last mile polish and fixing in Airplane is going to be pretty easy to do, because it's spitting out something that you've seen a thousand times before.

Whereas it’s just going to be really, really hard to do if it's spitting out like a Retool app and you're like, "Well, it looks correct, but I'm not really sure. I have to click on 800 things to really audit what the heck's going under the hood."

So, already as a pain point, I think it'll be just hypercharged. I think we're very well positioned for that kind of shift.

If everything goes right over the next 5 years, what does Airplane become? How does the world change as a result?

Today, with Airplane, we have the set of building blocks: tasks, views, workflows, etc., that lets you create powerful internal tools with a lot less work than you would've had to otherwise. You write a lot less code, but still have full extensibility.

The fundamental concept stays the same for a while. Probably we do interesting things like storage and other building blocks that further expand the variety of apps that can be built in Airplane. What will happen with both Airplane and the world is that software is going to get significantly easier to write due to things like LLMs. That will dovetail really nicely with the way Airplane is built. Airplane is a code-based platform. As code gets easier to write and create, something like Airplane will get exponentially easier to spin up.

Right now, if it takes you 20 hours to build the app that you want from scratch, maybe it takes you three hours to do so in Airplane. Five years from now, it'll be a world where that same thing will take you five minutes in a combination of Airplane plus LLMs and things like that.

You'll get a world where when app creation is that easy, where building internal tools and things like that is that simple, you'll see things like Airplane fill more and more of the gaps where off-the-shelf SaaS doesn't work. You'll also see where people are like, "Well, I could procure off-the-shelf SaaS for this, or I could just build a slightly more bespoke version of it than I need using Airplane" because of how easy creation will become. You'll actually see things like Airplane replace a lot more dedicated SaaS vendors because you'll be, "Well, I could buy that thing," or "I could just not buy that thing and use this platform I already have, build something in there that's actually a better fit for my needs anyways."

That's the thing I might see in five years but I don't know. The world's about to get really weird, so it's hard for me to predict.

Disclaimers