SACRA

EXPERT INTERVIEW

UPDATED
06/08/2025

Ayan Barua, CEO of Ampersand, on infra for AI agent integrations

TEAM

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

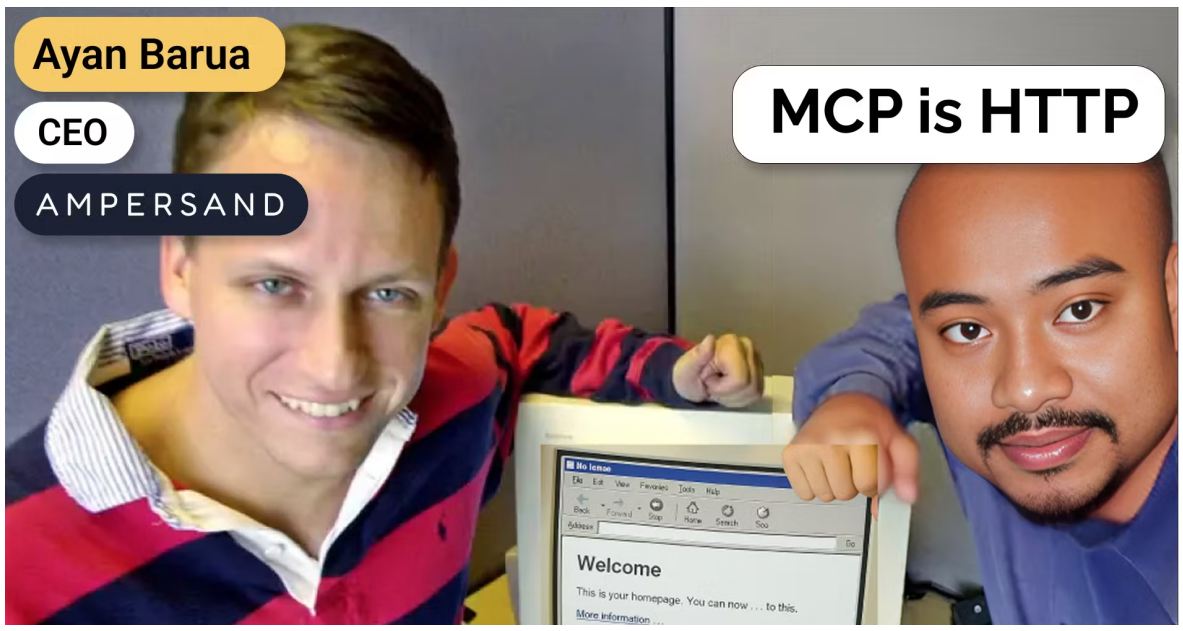
DISCLAIMERS

SACRA

Published on **Jun 08th, 2025**

Ayan Barua, CEO of Ampersand, on infra for AI agent integrations

By **Jan-Erik Asplund**



Background

Last October, we interviewed Ayan Barua, co-founder & CEO of Ampersand ($4.7M seed, Matrix) about the history of iPaaS and the rising trend of native product integrations.

We reached back out to Ayan to talk about how MCP and agents are changing how SaaS companies think about integrations.

Key points from our conversation:

- **SaaS in its current dashboard—and workflow-centric form—built on the 2010s UX pattern of buttons, views, and user-driven actions—is being eclipsed by AI agents, with buyers of software increasingly expecting ChatGPT-style systems that can integrate & reason over their data across all their tools via existing APIs, synthesize it, and drive outcomes.** “You cannot throw a DIY SaaS—which is basically a combination of dashboard, workflows, and integrations, where they have to figure everything out themselves or hire consultants to make sense of it—at a B2B buyer. Buyers don't

want to learn your software. They just want to get the job done. Business software is all about getting the job done. AI Agents and agentic workflows—systems that can reason for you and help you more than the original SaaS premise—are what people are demanding more and more."

- **While unified APIs promised fast integration for human engineers between your SaaS app and entire categories of tools like sales & CRM (Salesforce/HubSpot), HRIS (Workday), and ERP (NetSuite) by flattening 3rd-party systems into the most common fields via a single integration point, AI agents threaten to render this abstraction unnecessary, as they can go deep—parsing arbitrary data structures, mapping to tenant-specific fields, and covering all edge cases—by default.** "When a bunch of these unified APIs started, their take was, Don't do complex things. Shove complexity out. Here is a simple model, map to it, and you can get started.… In an AI-native world, the concept of unified APIs makes no sense because by definition, I am able to understand a customer tenant 10x better than two years back. Deep is the new shallow because AI is much better."
- **As Ampersand, Finch, Merge, and others all move to launch MCP servers and agent-facing SDKs betting MCP will become the HTTP of agentic SaaS, they're aligning around a shared protocol that gives AI agents access to 3rd-party systems via declarative "tools" for reading and writing data layered over existing APIs—enabling agents to retrieve the live customer context they need to reduce hallucinations and take the best actions.** "APIs are built for developers to programmatically take actions on systems. I'm a developer building a CRM integration, coding my thing, working on the API… MCP is a protocol designed around the agent talking to a system. It's a foundational shift that puts the LLM and the agent at the center, not the developer."

Interview

Last time we talked in October 2024, you mentioned AI agents, but we did not go into it deeply. What has changed in SaaS since we last spoke, both from a SaaS developer perspective and from an end customer perspective around demand for AI and agentic solutions?

I actually have only one answer to this. Every new B2B SaaS will start out as an AI agent and/or an agentic application. In 2025, if you look at legitimate companies with good entrepreneurs coming out of their previous successful exits, they're all starting new agent companies.

SaaS in its current form is likely dead. The reason is that buyers of B2B SaaS are demanding more from their vendors (and a lot of that is stemming from how consumers use ChatGPT). You cannot throw a DIY SaaS, which is basically a combination of dashboard, workflows, and integrations, where they have to figure everything out themselves or hire consultants to make sense of it, at a B2B buyer. Buyers don't want to learn your software. They just want to get the job done. Business software is all about getting the job done.

AI Agents and agentic workflows—systems that can reason for you and help you more than the original SaaS premise—are what people are demanding more and more. That shift has accelerated. The last time we chatted, these were just popping up here and there. That's how companies are starting now. And it's been only six months.

In terms of agent B2B SaaS, we're seeing (1) agent-first workflows like Sierra built on top of existing systems of record like Zendesk and Salesforce, (2) BPOs like Decagon meant to replace outsourced support teams and (3) agent UIs built on top of existing tools like Intercom has done. What's your view on the categories of agentic B2B SaaS, where do you see the demand and how do you see these categories converging or diverging over time?

These are the same outcome overall—customers and users of B2B SaaS want to do less grunt work. They want to get to outcomes—faster, better, cheaper. (1), (2), and (3) are all different methods to achieve that. There may be additional methods, and these methods will depend on the type of outcome and the type of buyer or user driving that outcome.

If the method is to do customer support for B2C companies like Sonos, and Sierra is quite B2B2C—then you'll see autonomous CS agents (from Sierra) talking directly with the B2C customer (of Sonos). They will likely delegate to a human after a point. If it's a higher value contract where you're almost replacing the BPO, then it could be more high-touch. Humans

aren't going away—they're helping accelerate the outcome much quicker. In high-value use cases, we'll have agents and people working together.
Fin is also interesting because it's Intercom's newest iteration. It shows how even older generation of SaaS is being repositioned as agents. It's very convergent. Whether you look at Sierra, Decagon—new companies with new stacks—or Intercom, which has been around for ten years, it's all convergent.

The convergence is around the fact that SaaS had a lot of investment going into humans using it and building muscle memory of how to use it. Go into this dashboard, click on this thing, scroll down, do this thing. There's a lot of "app mode" UX that has been built out over ten to twelve years. But that mode may not be the most efficient mode because buyers want a voice prompt interface. They want to chat with it the way they chat with o3. The UX that we build out for the next twenty years may not be as relevant as app mode. Directionally, it's all going there.

Talk to us about what AI agents can do with unstructured data that couldn’t exist before—for instance, what an AI support agent can do if the support SaaS has an integration with call data from Gong that wouldn’t have been possible before.

I actually wrote a post about this. We did a partnership with Gong recently, about a month back. We're natively able to bring a lot of Gong data into these platforms and also push it back into Gong if needed.

What Gong gives you is the unstructured intangibles in addition to the structured tangibles. CRM is a relational database at its core. There are some semi-structured fields and attachments, transcripts, but CRM is largely a relational database that you can extend with custom fields and custom objects per tenant configuration.

Gong, on the other hand, gives you more intangibles. You can almost pick up that there was a twitch in the eye of the prospect, and you know you lost the deal at that moment. That's not going to make it into CRM the way it's currently set up.

If you zoom out, there are essentially three types of data: structured data, semi-structured data, and unstructured data. The customer of a software vendor has all three of those datasets across three different environments. Some is in CRM, some is in Gong, some is in Fathom, some is in Google Docs or Notion. That context is distributed across multiple systems, sometimes by design, because one system, like Google Docs, can do what CRM cannot—like live editing and collaboration.

Someone like Gong is now owning large parts of the go-to-market stack because they're exceedingly good at handling unstructured data. They're also building agents—revenue agents. It's the same convergence of old SaaS and new SaaS—what I just described with Intercom.

Tell us about the problem of legacy API 'pipes' and infrastructure in building agentic experiences.

APIs are built for developers to take actions on systems programmatically. I'm a developer building a CRM integration, coding my thing, working on the API. But LLMs are not developers. Agents are essentially not developers. They're essentially a biological computer with some human-like reasoning characteristics, but they're not humans.

MCP is an attempt at an AI-native protocol and many protocols have impacted how we function as a society, such as HTTPS, TCP, gRPC, GraphQL, etc.. All the rest of the protocols are essentially developer-centric so that developers can build systems that talk to systems. MCP is a protocol designed around the agent talking to a system. It's a foundational shift that puts the LLM and the agent at the center, not the developer.

The good part here is that it opens up a bunch of new use cases where nontechnical people who are somewhat technical but aren't in the code can also voice prompt these systems very skillfully. We're going from a programmatic API-driven world to a more nondeterministic agent and voice prompting-driven world. MCP is a good protocol for that.

Assuming the community momentum that MCP has, it can really make it the de facto standard—though you never know if others could block the protocol—the reason this got so popular is because it puts the agent workflow at the center and doesn't

try to reverse engineer from the standpoint of the API. That's a foundational shift.
I wrote an article recently about what MCP does and what MCP doesn't do. If your CEO tells their developer to “go use MCP to solve all their problems”, it's like telling the developer “go use HTTP and it'll solve all your problems”. That makes no sense. Like any other protocol, MCP requires a ton of orchestration, and there are certain aspects that MCP excels in. That's where the community momentum is.

MCP promises to be a USB-C port for AI-agent context. Does MCP become the default way to get LLMs the context that they need? What are the strengths and limitations of MCP?

At the core of MCP, is the first serious attempt at establishing some kind of common language that AI agents learn to speak in. Think of it as providing a common walkie-talkie channel, allowing them to request context, perform actions, and stream results without needing to invent a new dialect each time.

An agent calls separate tools—in the B2B context, it's SaaS provider data, whether they're internal or external APIs, databases, file systems—just by talking through MCP. It's not talking 20 different dialects to 100 different systems. It’s just speaking a gateway language.

Then there are orchestration platforms like Ampersand that operate on top of the MCP protocol and provide AI agents with capabilities that MCP deliberately avoids. Things like API authentication and token refreshes, object and field mapping, privacy, security, governance, rate limiting, retry logic, error handling, etc. There's a bunch of difficult infra work you still need to do for your agent.

For agents, is latency a bigger problem or having the right context, a combination of the two?

There are multiple classes of problems for agents. It's a gateway to different systems—it's a protocol. MCP is not your AI agent brain. How do you feed your agent brain real-time data context? That's the big question here.

How do you orchestrate in and around MCP securely? How do you do it in a way where there's no tenancy problem? If the

agent is an agentic application that must deal with a hundred different Salesforce tenants across a hundred different customers, it cannot simply grab data from one tenant and mix it with data from another tenant. That would be a privacy and security nightmare for the agentic app builder.
Tenancy management is one of them. MCP is not your security department. MCP is not an ETL. MCP doesn't take care of data lineage. MCP is more synchronous than async, yet. MCP is not your bidirectional sync engine. MCP is neither cost-aware nor rate limit-aware. MCP is also not a universal data model—you still have to think about your own schema. MCP doesn't have an error handling framework, the way you would want to build a robust error handling framework. MCP lacks a robust consent and governance layer.

I'm not complaining here. These are all things MCP shouldn't be doing. This is like HTTPS needing OAuth. There are other tools that will complement MCP.

MCP doesn't have native observability. MCP doesn't have versions or compatibility. There are multiple such challenges that others will build solutions for. I can't even say that's a hole—these are things that the protocol shouldn't be responsible for. Others should be adding and complementing MCP with their tools.

MCP needs dev, test, and sandbox environments. When you're taking autonomous actions, you need to understand how risk is propagated across systems. MCP shouldn't do that. Other developers and systems will take care of problems like these.

What are the incentives for Zendesk or Salesforce to build better API infrastructure for integrations if it enables them to extract an agentic tax in the form of a higher volume of API calls (e.g., polling) and enable them to compete better with their own vertically integrated agent UI on their SaaS?

I would argue that they are already investing a lot in their APIs. People often discuss the limitations of Salesforce APIs, but Salesforce is a complex product with a vast surface area, and its APIs reflect that complexity. Developers building with Salesforce CRM APIs for the first time get lost often. If you land on a huge continent and don't know where the map is, you're going to struggle.

For companies like Salesforce, HubSpot, and Zendesk, which invest in their APIs, it is evident that newer Agent companies are generating or writing back numerous insights into these Systems of Record. This lets new use cases to open up. Sierra, on top of Salesforce, would likely increase the amount of data that is being sent to Salesforce. So now Salesforce becomes a stickier Systems of Record database because of Agents

There are reasons for incumbents to actually want more API usage. The data is getting back into the their Systems of Record.

I think that the B2B saas future will be a little more vertical. There are untapped places and opportunities where you can almost leapfrog ten years of construction equipment movement and SaaS around that. ServiceTitan broke out in the last five to six years because it was so vertical.

A lot of these platforms will want to build agents themselves for very horizontal use cases. Then there are agents that others will build in and around these systems, not necessarily just one system. They're probably focused on multiple systems of record. Get data from a communication platform like Gmail or Exchange, marry it with another communication platform like Slack, then marry it with Gong, and use it to orchestrate in and around ServiceTitan.

Those types of vertical agents, the horizontal platforms will not build. The market may fragment with Zendesk building five agents that are core to the utilization of Zendesk itself, and then the rest of the agents are basically these other marketplace agents—other SaaS companies that are customer support adjacent, but not really. They're either verticalized or maybe a little outside the realms of what Zendesk or Salesforce might do.

That's how I think about it—it's the next iteration of business software. Systems of record and systems of intelligence aren't going away. All of them are getting 10x smarter.

Slack recently announced that they're locking down their API or making their API more expensive to use, similar to what Twitter did a while ago. It seems likely that there might be more companies that will do this—hoard the data

and make it really expensive so that you favor using their solution. Can you play out why that's a bad strategy and break that down a little bit?
When the Internet first came in, my parents were like, "Don't go on the Internet, man. Don't do this. There's some bad stuff that's going to happen." This is the 1994 moment of Internet for AI.

Build more open ecosystems, and they will get enriched along the process. Some people will close it down further, but the data at the end of the day—if I'm GE and I'm spending $10 million on Salesforce—it's not Salesforce data. It's my data. This call is being recorded in Fathom. Who does this call belong to? You, me, or Fathom?

The customer will win at the end of the day because they're paying a gazillion dollars to get to a set of outcomes. The systems of record like Salesforce, HubSpot, Microsoft, SAP, and Oracle have a lot of trust built up with customers, and that's their moat. But customers want to move better. Customers want to move faster. Customers want more value creation with less money.

If I'm able to ship 10 times faster because of Cursor—and that's already established, it's not some random wish somebody had—then at the end of the day, in business software, customers have the last say. Wherever customers want to operate and wherever they want to take the data, they will take the data.

In this moment, you can either play defensive or you can play offensive. Offensive is essentially: you want more API access? I'll give you even more API access. I'll make you so sticky that you continue to trust that I'm going to be an amazing steward of data.

When you're talking to developers who are building stuff with agents, what are some of the specific problems you see them coming to you with and what role is Ampersand serving for folks right now?

A lot of our customers are AI companies—AI-first companies starting in the last few years. The reason they come to us is similar to what I mentioned earlier. They believe that for their agent to function really well, they need all the enterprise

context from their customers, which roughly translates into deep CRM or go-to-market stack integration and deep ERP integration.
They believe they don't want to invest twelve to twenty-four months building it themselves. They want to trust or use things like Ampersand to have the flexibility and configurability that they would have if they built it natively, but without having to build it natively. They're leapfrogging six to twelve months of their dev lifecycle, and they're not having to maintain it at all.

Our pitch to them is: ship amazing AI, not integration plumbing. What you do with the ingredients will define whether you're Michelin star, but we will get you the right ingredients. You want the best tuna on the West Coast, we'll get you that. But how you roll the sushi is your secret sauce.

In a world where things are moving extremely rapidly and velocity is key, we are basically an engine for that velocity to stay with them without them having to get burned by integration plumbing.

We talked a lot about the strength of native integrations versus universal APIs or middleware last time. Is that an even bigger issue with agents? And in what use cases does it become more important to have native integrations?

Humans are generally not looking forward to doing object and field mapping. Let's say I'm a Revenue Ops person bringing in this new Sales tool because it's a very good tool, and I want to roll it out in my company. I'm not really looking forward to having mutliple two-hour calls with the solutions engineer where they're like, "Can you tell me about *pipeline_stage_c*?" And I'm like, "Dude, it's pipeline stage. It's a custom object." And then the person is like, "Cazn you also talk about opportunities_stage_c?" I'm like, "I just told you, man. It's also a custom object for the opportunity stage."

The object and field mapping that a native integration gives you is table stakes, but it's still extremely painful for people to go and map those things. This is at the heart of a native integration. I have a customer, and they've set up Salesforce, Workday, and NetSuite in a certain way. They have a complex schema. I need to understand their tenant's data model and then map it to my schema. I'm doing that for a thousand

customers, or hundred of customers, or whatever number of customers I have.
Ampersand's exercise was to basically make that one-to-many mapping extremely easy because if you deconstruct what a native integration is, that's what it does well. The native integration is able to go to attributes and fields and custom objects unboundedly.

The second order problem is that once you give that optionality, people still have to drag their feet to go and map those things. AI can actually do a fantastic job of this.

In an AI-native world, the concept of unified APIs makes no sense because by definition, I am able to understand a customer tenant 10x better than two years back. When a bunch of these unified APIs started, their take was, "Don't do complex things. Shove complexity out. Here is a simple model, map to it, and you can get started."

Now, if you look at the unified model, it is MCP in many ways. If you take MCP and add some vertical focus on MCP, that's going to be our AI SDK. We're saying, okay, you take MCP. Here is Ampersand AI SDK. You take messaging buses, put them together, and that's for all your go-to-market stack integrations—native and deep.

Most companies will go to a point where integration providers have to understand that everybody will have native integrations because native integrations are much easier to do. If I can do a native integration in two days instead of two years, I would want that. The directional compass of data exchange between buyers of software and sellers of software will be deep by definition.

Deep is the new shallow because AI is much better. If you chat with o3, it knows you better than many people you've known for twenty years.

A big driver for integration middleware has been the explosion in the long tail of SaaS. Do agents result in a rebundling cycle where most of the long tail of SaaS with minor differences in UI, data structures and pricing effectively die off?

You have to understand why SaaS became fragmented in the first place. Look at my background—I was VPE at G2. Before that, I started Siftry. Before that, I started Credii. These were all trying to bring sanity to the SaaS fragmentation problem, whether it's discovery or spend management or compliance around it. Along the way, I realized there's no infrastructure that is stitching SaaS together in a fundamental way. So Ampersand is one attempt to solve fragmentation at the infrastructure layer.

The reason why SaaS got so fragmented and why there are so many tools is because it's catering to different workflows in the enterprise. It's catering to different people with different needs in the enterprise. The reason why there are so many call recorders is because every one of them has created new nuances of the call recording software stack.

SaaS will continue to be fragmented because SaaS gets verticalized pretty quickly. It has its own niche, and then it'll continue to fragment. An agent is basically agent-first SaaS now in many ways. You'll see more people building these agents. I don't think suddenly the SaaS market will shrink to 2,000 legitimate SaaS companies instead of 200,000. That won't happen.

But what will happen is that human beings will consume less SaaS through traditional UX. I don't want to go to my twenty-fifth dashboard and do a bunch of things there. It's too much cognitive overload. I want to have discussions like this, not spend time on my twenty-fifth dashboard.

The workflows and the databases will remain. The UX layer could be something more agentic, where dashboards are essentially exposed as insights through MCP servers, and there are agents orchestrating. ChaGPT and HubSpot native integration is a good example.

I don't want to go to my accounting software. I need it really badly because otherwise the system is a mess, but I'm not super excited to go and look at transaction level details. You may have a transaction agent that is categorizing all types of transactions on the ledger. What I want as a business is good clarity on how systems are functioning, how different sides of my business are functioning, revenue recognition—how are we doing revenue recognition?

There might be a revenue recognition agent that gives me a good overview, maybe in simple dashboards, or maybe in voice, and maybe in email. A lot of the SaaS UX—you may end up investing way less in building a beautiful UX that nobody will use because folks are busy voice prompting, not clicking.

If agent SaaS increasingly prices based on outcomes, how does that flow downstream and affect how an infrastructure product like Ampersand prices?

Ampersand is not the first integration company, so we carefully studied the pricing models of all previous generation integration companies: task-based, user-based, connector-based, connection-based, etc. Eventually, we felt that there is a network, storage, and compute cost of data movement. To run and operate this multitenant data movement platform, we have our costs. We're trying to align our pricing model with that infrastructure consumption. That's how we think of ourselves—as a middleware infrastructure company.

The consumption model is also hitting the application layer strongly. The application layer was previously seat-based. I have a seat at the table for using this software. Whether I use it or not doesn't matter. I will have to pay just for that seat.

In many cases, that still makes sense. In many cases, you will see that model being extended to seats—maybe human seats, plus agent seats, plus agent actions. Salesforce is trying different things with Agentforce pricing. They're at least attempting different things and seeing what sticks, what doesn't, because nobody really knows what the SaaS future is going to look like.

Seat-based gives you a lot of predictability. So people like it. I'm $10 per month per user. Makes sense. It's easier—no cognitive overload. AI has actions. If I charge $2 per action and then offer premium actions at $5 per action, that becomes complex.

Right now, the shift is going to be a little gradual because the world operated on a simple pricing metric like user seat-based pricing for ten to fifteen years, maybe twenty years. Now it's like, okay, what does consumption mean in this world? There will be many attempts at getting different pricing models going, but at the end of the day, companies value ROI more than the

expense. If I'm putting in $100k and I can make $300k back shortly, that’s a tangible output and outcome...
The application layer has to figure out a lot more about how to actually do outcome-based pricing. It's a very complex question, Jan. I don't have the best answer for how you arrive at a metric that is universally adopted. But SaaS seat-based pricing is no longer the de facto go-to anymore, which is a big change in itself. It's an ongoing conversation.

With Ampersand, you set out to solve a recurring problem for developers and a pain that you personally experienced. With LLMs and agents increasingly choosing the infrastructure tooling and writing the code, how does that affect your personal view of and investment in the problem? Is infrastructure a place where developer “taste” is more important than ever, where developers need to spend more time, or does AI buying and coding threaten to commoditize tools like Ampersand?

If you deconstruct developer taste, what that means essentially is reducing barriers to adoption. Developers are context switching all the time. There are many things they're doing every day. If they're already doing 14 things, and I put in the 15th thing on their plate, and if the 15th thing has a poor introductory curve, then developers are like, "This thing is not good. This isn’t good software."

The developer taste you're talking about is basically ergonomics and abstractions that make a complex problem easy to solve. That doesn't go away. A lot of the Agent taste will be a direct reflection of how developers orchestrate those agents because the agents are still going to be focused on a certain set of tasks. The tasks will be defined by the people who use those agents.

In our case, whether you're vibe coding or building a more serious application, those use cases are convergent. You still need to get data from your customer's environment. So Ampersand's mission hasn't changed. Ampersand's mission is to enable product builders and developers to really build connected business software experiences.

The mission is even more interesting because now we have two consumers. The first one is the developer. The second one

is the agent of the developer. The agent of the developer is working in tandem with the developer's goals.
The reason we were pretty quick to adopt MCP—the first thing we did was launch an MCP server for our docs. So you could say, "Hey, I want to subscribe to HubSpot contact changes of all my customers. Can you build an Ampersand YAML for that?" Developers can now code their way into a complex integration super-fast. They are in Cursor. They have our MCP server, and they're basically saying, "I need this. I need that" And the outcome is a very good integration manifest, that is on top of the MCP server that we built for our docs.

We are now launching a proper MCP server plus an AI SDK so that agents can directly use Ampersand to call tools. Basically, AI agents can perform CRUD-like operations on connected SaaS tools through Ampersand. This is a common set of verbs that developers use to spin up Ampersand orchestration across multiple systems.

To answer your question, Ampersand will be developer-facing and developer agent-facing, but with the common goal that you're building connected experiences with your customers.

Snowflake and data warehouses are obvious winners from the trend of AI agents. Second, there appear to be winners emerging on the customer-facing side like Sierra that are potential Salesforce disruptors. How do you see value aggregating at the data / integration middleware layer and why will Ampersand be a big company and major enabler of agentic SaaS over the next 5 years?

Storage—let's talk about Snowflake and Databricks. Storage without fresh bi-directional context is more of a cold archive. AI agents do much better than the previous generation of software one of thing: velocity. They can produce a reasonable set of actions at a much faster pace.

If you do deep research, for example, with a deep research agent, the amount of research that it can do—a McKinsey consultant would probably take three months whose only job is to just research. AI is producing value 10x, 100x, 1000x faster. Which means that agents need a lot more fresh data. **Agents need real-time context**. If something changes in customer systems or other environments, they need that change to

propagate to them much quicker. If they don't have these reliable sets of changes, they could hallucinate more.
The middle layer guarantees real-time reads and writes. It guarantees real-time change data capture. The data exchange part becomes even more important because the storage layer, currently modelled around batch ETL and nightly jobs, reports showing up the next day is too slow. Agents will not wait for twelve hours or twenty-four hours. In some cases, they want to ship things right now.

The network, storage, compute, and the middleware layer will evolve into more of a real-time messaging infrastructure between complex autonomous systems. Every part of the B2B SaaS stack is up for grabs, by the way. When you say Snowflake and Databricks—great companies—but if this is the 1994 moment of AI, then all parts of the stack are up for grabs. You may see middleware companies actually become the storage layer as well.

I actually don't know how it's going to play out, but all parts of the enterprise stack are fair game right now. And every CEO, incumbent or new, knows it.

Disclaimers