EPISODE 1157

[INTRODUCTION]

[00:00:00] JM: The Salesforce ecosystem has thousands of developers, designers, product people and entrepreneurs engaging with each other. Salesforce exposes APIs and SDKs that allow people to build infrastructure on top of the Salesforce platform. In a previous episode we explored how the ecosystem works as a whole. In today's show, Chuck Liddell joins the show to talk about how developers themselves engage within Salesforce. Chuck is the CEO of Valence, a Salesforce app exchange ISV that adds native integration middleware to Salesforce. So he's a deep expert in the Salesforce ecosystem. I hope you enjoy the show. And full disclosure; Salesforce is a sponsor of the show.

[INTERVIEW]

[00:00:45 JM: Chuck, welcome to the show.

[00:00:46] CL: Great to be here.

[00:00:47] JM: We're talking today about Salesforce and building on top of Salesforce. So I think it's fair to ask at first, most people think of Salesforce as just a big platform for doing your sales and marketing on a cloud platform, but it actually is this big ecosystem where people are developing third-party applications and there are lots of developers that work entirely in Salesforce just like people would think of WordPress developers. Tell me a little bit about the Salesforce developer ecosystem.

[00:01:19] CL: I think WordPress is actually a pretty rich analogy there, because it's kind of a business application construction tool, right? When I first got into it, I really thought of it as a CRM and I was doing things to support sales teams. And those are the types of projects that it originally was doing. But they've really expanded the sort of surface area of the platform over the years, and it's become a much more general purpose building tool. I really think of it now as more of like an Azure and AWS where if I just want to build apps, it's a good place to do so.

[00:01:53] JM: What would be an example that might surprise people? Because they think of Salesforce as a sales and marketing platform. What would be an example of something more abstract or off the beaten path?

[00:02:04] CL: So I've done a few different interesting projects in the past, and I'll describe a couple just to give you a sense of what's possible. We did a build for a company that sells used laboratory equipment, like big manufacturing equipment. They sell secondhand equipment, and we built sort of a set of apps for them where there was a mobile piece. A technician would go to a factory that's been closed down, would take pictures of equipment and sort of inventory things. And we would track the inventory in Salesforce, items that could be sort of processed and put up for sale. And then we did a mash-up with S3 to store all the images, all the pictures that have been taken of the inventory and we would affiliate the images in S3 with the item metadata in Salesforce. So it's all sort of an affiliated set of records.

And then we actually did a storefront where there's a public website that people can go to to see this inventory and it's searchable, filterable. You can register your intent to purchase something and be contacted. That was all sort of affiliated with this business, but it was running on Salesforce. Sort of full stack Salesforce centric. So that's one example. Another example we did is there was a project some years ago where it was a lead marketplace where these teams wanted real-time access to leads. You would register your interest on a website or something and you'd get a phone call within a few minutes. And so there was a very dynamic nature to it, and what we ended up doing is sort of building a multitenant environment on top of a single Salesforce org where this company that we were doing the project for would sell access to their system, their engine of sort of lead processing and validation. It's all about sort of good quality leads and filtering and evaluating the leads in real-time.

And so we built a rules engine that could sort of process these leads that they were selling to their customers and we tied it to Elasticsearch, which has phenomenal sort of real-time filtering and querying abilities, and builds essentially like an analytics dashboard using Bootstrap where people could build their own analytics, like these end customers, the customers of our customer could go into a dashboard and can figure little cards with all kinds of interesting metrics that they want. This value divided by that value over this period of time, really kind of fairly arbitrary calculations. And it would generate a sort of a sanitized live query against the Elasticsearch

instance off platform. That was sort of synced up with the Salesforce data so that we could both collect the data in Salesforce and allow them to interact with it in a very dynamic way, which was really interesting. So those are two examples of some projects that we've done that are maybe not quite the norm.

[00:04:49] JM: Tell me more about the abstractions that Salesforce exposes to the developer for building applications.

[00:04:56] CL: So one of the things that really intrigues me and delights me about the platform, I'm a little bit of a wonder, a purist when it comes to sort of software patterns and good design principles. I'm a big fan of Eric Evans and domain-driven design. And one of the things that Eric Evans talks about is this idea of ubiquitous language, that there is a set of nouns and verbs that make sense to the business people and the architects, the developers of a system and that those sort of natural language items that everyone understands with the definition of drive both the way the business works and the way the technology works. I think it's a really interesting idea and it's important.

And I think more than most systems I've interacted with, Salesforce has a really good job of making it possible to design a system where ubiquitous language is in the forefront. What I mean by that is it's such a metadata-focused platform. You can define custom objects, which are tables, and fields, and sort of processes and flows and just name them and have them do what you want. You can stitch them together and build relationships between things. And because everything is so tied to that metadata, just by creating a new custom object, it becomes surfaced in the API, it becomes available to you in your declarative tools. It becomes accessible using sort of compiler time checks from your code. So you get to just sort of create all this abstract conceptual stuff that becomes very accessible to all the tools with no effort in your part.

So it gives you the flexibility to really tie the technical work you do to sort of the abstract ideas, which I find really interesting. This is probably one of the most successful things they've done with the platform in my mind from a design perspective.

[00:06:44] JM: Let's talk a little bit about the security model. So the user security model for building on top of Salesforce, what does that look like?

[00:06:53] CL: So there's every access to the system. It's affiliated with a user record, and the user record doesn't have to be a human being, but there's always a user record tied to all kinds of access. So any time a record is created, touched, it's done from a user perspective. So you have that, and then you also have the idea of an org, right? So an org, or an instance is what Salesforce calls a single slice of their multitenant environment. So as a business, you have a single org and all the records in your dataset that you interact with belong to your org. And actually under the hood, it's a much larger database with different orgs coexisting on the same pod. But from your perspective, you only access stuff that's tied to your org and you have no controller over that. You can't get outside of your org. It's very well sort of firewalled from the other customers.

And then within that model, a user can be affiliated with different sets of permissions. So there's sort of large control called a profile, and a profile is sort of your major, what you are, like an admin or marketing team or sales team, that sort of thing. And then within profiles, you can set all kinds of access rules. It's just sort of a big bucket, a big categorization. And then much more granularly, there's this idea of a permissions set, which is just a collection of things that someone is allowed to do. You can give it whatever name you want. You can create as many as you want, assign them to people. And those are meant to do sort of enhancements, right? It's all sort of progressive enhancement. So by default, you want people to know access, and then you can add on to it. So profile usually gives a minimum basic set of things and you further enhance that with permission sets.

Every record in Salesforce has this idea of an owner. So every record has an owner. And the permissions try from that owner. So if I own a record and we work together and you're my boss, then permissions can be granted so that all the records that I own my boss also can see, or make my boss also can edit, or maybe my team has access to. There's actually a really robust set of ways to do the permissions. It can be overwhelming for some people, for sure. So actually when you're first getting started on the admin side and trying to understand how to model the stuff, because it's so flexible, but it's really powerful. I don't think I've ever run across a situation that couldn't be modeled at all bu ways that end up being a little complicated.

So I'll give you an example. This thing is a little more complicated, is this idea of public group, which is just a collection of people, just a bucket. And so you can define – If you have no sort of natural affiliation for the user that's like the hierarchy of the org or something, but there is some way that people were affiliated, maybe cross-functional teams across different departments or whatever. You can just put them in a public group, sort of a last resort. Just a way to put people together, and you can affiliate commissions of the public group. So you can say, "The blue team has these sets of permissions and these are the criteria for being on the blue team." But often you need some sort of programmatic level of definition. So you can really get in the weeds with the permission system in two ways. One, the public groups are actually sort of metadata that you can manipulate using code. So you can assign a person to a group or remove a person from a code. So you've built sort of complex permission system of people in the past where really nothing else worked, and we had sort of the last resort of using public groups. And kind of dynamically put people in groups and took people out of groups.

So let's say for example you have people register to participate in something internally, right? They're on a project. You might have a group affiliated with that project. And when they sign up or be part of the team, they get added to this group. And when they're removed from the team or the project ends, they're removed from the group. You can do sort of programmatically. And then if you really want to go even one layer deeper, you can actually have sort of programmatic sharing at the record level.

So you've got all these different layers of sharing, right? You can do profile or permission set. There's a lot of rules-based sharing too. So if the owners this or if the value is greater than a million or if they are in the state of Utah, you get access to this record. There are all kinds rules-based sharing. And at the end of the day, if none of that works, you can fall back on programmatic sharing and say, "Okay, there's no sort declarative way I can define that they get access to this record, but they really should have access. So I'm going to do it programmatically. I'm just going to write a row to a certain table using my Apex programming language that says, "This user or this group has access to this record." And you can build very rich permissions systems on top of that.

At the end of the day, it's really just a rules engine. It's trying to compute grants, right? Which user or group has access to which record and what level of access? Is it read? Read/write or

ownership sort of transfer level access? And this is maybe 20 different ways you can define the rules. But at the end of the day you're just competing that simple grant record. And in fact, Salesforce pre-calculates those records. So they are constantly generating these tables, these grant tables of who has access to what records so that despite the complexity of the calculations for who has access to records, the reads of course are much faster because it's a very simple lookup table.

[00:11:45] JM: You've mentioned the programmatic layer called Apex a couple times. Could you go a little bit deeper into what that is?

[00:11:52] CL: So Apex is Salesforce's proprietary programming language. It's not open source, but it's very well-documented. It's very similar to a C++ or Java object-oriented similar syntax. Under the hood, it is Java with a bunch of other stuff built on top of it. But from our perspective kind of working at the top layer, it looks a little different than that. It's a great language. I've actually really enjoyed working with it. It's got some weird little sort of aspects to what it is, because I think of where it came from. This is me speculating a little bit. I don't have the full story on the origins, but my understanding is that when Salesforce kind of first got started, Apex was really meant to be just a way to get a little extra logic on top of a database transaction. So you write to the database and you do a little extra manipulation or that sort of thing. And it's really grown over time into a much richer sort of standalone programming language on this platform. But some of the side effects of that still exist, right? Which is it was always meant to be I think sort of a small enhancement here and there and not really like standalone complex projects. So people build crazy things in Apex. They build tens and tens and thousands of lines of code written in Apex that do all kinds of complicated things. And the organizational structure isn't totally there for Apex, right?

So for example, to be more clear, there's no idea, no sense of packages or really a lot of folder structure in Apex. So everything is kind of in the same folder. And if you want to us or keep your classes separate, you have to name them in some way that's consistent. They're starting to make improvements to that to some extent. So one of the things that Salesforce has been working on recently is this developer experience where they try to make the tooling better and the practices better for developers. And they, as an example, recently allowed you to actually track your source code in a folder structure and then they will mutate it and flatten it when it

goes into the Salesforce org. So it's still flat in the organ and a little bit hard to kind of manage once you have hundreds of thousands of classes, but at least in your local development environment, it can be sort of split up.

So Apex is a pretty slick language. One of the nice things about it is that because it's completely specific to Salesforce, they have allowed syntactical sugar that's tied to their platform. So one of the really refreshing things for me when I first got into doing Salesforce work a while back was the database access, rights? So I'm used to – I came back from doing a lot of Java stuff with like MongoDB, or PHP, and MySQL and all kinds of random stuff, and it was always like set up the database connection, authenticate it, make sure it's secure, access it, have fallback plans if the database isn't available to you from your code. All that's gone in Apex, because it's built on the same stack.

So your Apex code just as a database query and you can assume that you got the results. You don't have to worry about connectivity to your database. You don't have to worry about authentication. And even more than that, because it's so tightly coupled, you have compile time access to metadata validation. So what I mean by that is let's say you have an account table that has a new field that an admin added called like account value or something, and if you refer to that field from Apex, it'll actually do a static type check for you at compile time. It will say, "Hey, you misspelled the field name." And conversely, if the admin goes in and deletes that field, but you're using it from Apex, they'll actually get a warning saying, "Hey, you can't delete this field. It's being referred to by code." There are some really interesting validations that are possible because of how tightly coupled Apex is with the CoreStack, but also some weird wrinkles from sort of how it got created.

[00:15:22] JM: In terms of actually building workflows or processes, there is the declarative layer. Can you talk more about what the declarative layer is and how you build applications at the declarative layer?

[00:15:33] CL: So in some circles, the Holy Grail of software is to not write code, right? It's a mission of a lot of different platforms and teams over time. And I'm a pragmatic person. I think that there's always going to be some combination of what you can do with code and what you can do with tools that are essentially writing code under the hood. And Salesforce really

believes in sort of a low-code/no-code approach. And can they create enough tooling so that smart people can do all kinds of interesting complicated applications without actually having to write any code?

So there are ton of tools on the platform to support this sort of initiative. There're workflows, which have been around for a long time. Basically sort of an if, this then that sort of behavior where you say, “Okay, if a record satisfies these criteria, then I want you to mutate it the following way, or I want you to send an email to someone about this record.” So it’s sort of the first one they did, and then they've added another set of them overtime. There are flows, which are much more robust. Sort of node-based behavior where you have like this, sort of if/else evaluation, and if true follow this path, if false, follow this path. And then manipulate records in the following way. Very similar to like a Node-RED, if anyone’s familiar with that. Sort of very sort of Flow-based behavior called Flow, naturally. And they also do something called process builder, which is very similar to Flow. Actually quite affiliated under the hood, similar technology. And that's a little bit friendlier, richer with metadata, not quite as powerful. So that’s just three examples. So sort of workflow-based logic.

But there’re actually a number of tools in sort of the declarative toolbox. You have things like validation rules where you can say, “Okay, I want to evaluate this record for these criteria,” and the criteria can be quite sophisticated. Salesforce actually has their own sort of formula language where you can evaluate all kinds of things about a record or make mutations to a record using their formula. And then they have metadata linkages back to the formula.

So this is going to be a drumbeat throughout our conversation, is all this metadata aware behavior. So for example, I can define essentially a reference table with information and maybe breakpoints for discounts based on volume, right? This many items sold. Is this kind of breakpoint? Or maybe it's really reference data. That's the names of all the states, or the names of some sort of information that we refer to a lot for our business.

And from, for example, a formula, I can pull on that reference data. Or from some of these declarative tools, I can pull on their reference data and use it as part of my valuation. So if this record is greater than breakpoint A, let’s do this logic. And I can define breakpoint A as an

abstract entity, right? I'm just going to call it breakpoint A. It's actually a metadata in a reference table somewhere else. So I can go back and update that reference.

Really, there's some very clean design here that is a common thread throughout all his different parts of the system. This idea that metadata is the most important thing. We're not going to repeat stuff as much as possible, and everything else is built around metadata and ties back to it.

So of course, you have all those declarative tools. They don't always get the job done. Unusual or complex things, typically you fall back to code. And then especially for speed, the coding on the platform is still much faster than most of the declarative stuff. That's gotten better over time, especially with the recent feature where you can actually run flows before save. But still, if you're doing things with a lot of records, it's typically better to write it with Apex still.

One of the things that I try to tell people when I'm teaching them about Salesforce, I do a lot of training for architects and developers, is that because it's such a large toolbox, you really have a lot of different ways to solve your problem. And a lot of them probably will look correct. And in fact some of them will be correct, but only a few of them are probably the most – Well, only one of them could be the most. But only a couple of them are probably really solutions to your problem.

And one of the reasons that people get tripped up is you won't necessarily know which one was the best for a long period of time and really with scale. That's one of the big Achilles heels for people that are getting used to doing the Salesforce work, is that things behave quite differently at large scale than they do at small-scale. So if you have a table that has 100 records and enter a thousand records, then fine. Like you kind of do whatever you want programmatically and declaratively around those records. If you have a table that has 10 million records in it or 50 million records in it or 100 million records in it and remember this very sophisticated security model around sharing, that's a lot of calculations around sharing, and you start to get into all kinds of weird little behaviors around, "Okay. Well, if you defined the linkages between these two records as this type of relationship, then it's going to change your sharing model in this following way." And when you scale that 250 million records X the number of users in your org, you start

to have huge numbers of sharing records that had to be recalculated if you make certain changes. So it really kind of takes you down a rabbit hole at large-scale.

[00:20:28] JM: And of course, every application needs to have some kind of UI layer. Tell me about the best practices for building UI layers in Salesforce applications.

[00:20:38] CL: So let's talk about UI layers. I want to come back to just the idea of layers in general with Salesforce. So of course, Salesforce is an evolving platform. Actually, one of things that I love about the platform is the how much it moves. It changes a lot. They have three releases a year. There's so much stuff in every release. It's actually really challenging, honestly, as a practitioner to keep abreast of all the information and all the changes and to actively learn all the time with the platform.

So what that means is that things change over time. There are all sorts of new ways to do things. So kind of back in the day, there was Visualforce, which is actually built on Java boldfaces. I don't know if anyone's ever been familiar with that library. That's not really well-known, but there is a little secret tip from back in the day.

[00:21:21] JM: There's definitely somebody in the audience who's listening who's familiar with that.

[00:21:26] CL: Well, you are special and you are in a small group of people that know that that's the origin of that features in Salesforce. So Visualforce has been around a long time. It's kind of a template system, very tightly coupled with the backend. Really pretty clean way to refer to fields and records. But getting a little long in the tooth, some years ago, Salesforce focused on sort of a Java-based approach. So Visualforce is kind of a thin client approach, right? It's a very thin page that renders server-side and you're sent just the HTML with a V state, right? So here are some data about the page session and then here's the actual rendered output.

Aura was a different approach. So, Aura was there library for doing sort of a JavaScript loading system where you have sort of this class-based widget module system in the browser. Kind of similar to an EXTJS sort of approach, and that was around for a while, pretty powerful. Definitely some interesting complexities with doing more sophisticated projects on it. Again, Salesforce,

they have this sort of – I don't know. I might be in an unusual group of people doing sort of complicated, very large projects with their tools. But a lot of the things that they build are designed to sort of enhance what they already have. They're only meant to be a huge standalone thing.

So Aura was challenged by struggling a little bit at large-scale. It's really good sort of module system, but it had performance issues. And so the sort of third iteration here and the most recent came out about a year and a half, two years ago, are called lightning web components, and that Salesforce is sort of interpretation of the web component standard. They've actually sort of participate in the group that guides that standard, and they've taken web components and essentially done the Salesforce thing to it and made it very metadata-focused.

So a lightning web component is a web component, but it runs on the Salesforce CoreStack. It has all kinds of sort of behaviors built into it. So they built a wire system where you could essentially have endpoints to pull data from and exchange data with that you really baked into the framework. And then they built a bunch of what they call base components, which are essentially building blocks for your UI. So you have a button, or an accordion, or various widgets the you can just grab and drop in and make part of your system. It's very object-oriented. ES6 module style, and they've actually open sourced it and put lighting web components out there for other people to use on any stack, which I think is a really interesting representation of character. And I'll explain that a little bit more.

Salesforce historically builds their own stuff, right? Apex proprietary language. All the stuff that you can use on their platform is not open source. They do some open-source stuff, but they're very much about sort of their walled garden, kind of an Apple style approach. And the lightning web components, being open source, of course the actual runtime on the Salesforce stack is not open source. They have all kinds extra security stuff built into it. But there is a version of the library that's open source, and its a philosophical difference. This is my opinion, right? I don't work for Salesforce. But I think it's a philosophical difference for them to really participate in the technology world in a different way than they have in the past. Rather than just build their own version of everything, they seem to really be thinking about how they fit into the larger sort of software ecosystem and what sort of libraries they want to help curate. What they want to use? What do they want to share?

So it’s really interesting to see them really participate in open source and be a little more open-minded and open arms with the rest of the tech world than they had been in the past. So anyways, I digress.

So we’re talking about lighting web components. So if you’re familiar with ES6 modules, these will be very familiar to you. They are extremely performant. They struggle a little bit because the web component standard isn't really totally done. So for example, Shadow DOM. We’re really getting in the weeds here a little bit. But Shadow DOM is essentially an internal encapsulated document object model for a single component in the UI. And they've essentially built their own version of Shadow DOM as a temporary placeholder until the Shadow DOM part of the spec is sort of fully realized out in the world as all browsers pick up on it and things get formalized.

And so there're weird little wrinkles with things like your CSS in the Shadow DOM with the lightning web component. And the team behind lighting web components is incredibly sharp. I really have enjoyed talking to them. And I think it's going to be a really interesting standard even more so than just in the Salesforce world. I think there is real potential here to go after larger types of projects. So their vision is that this is going to be a real framework, JavaScript framework for the web that people like and use and is fast and really nice to work with. They want to go out there and make a compelling alternative to React and Vue and Angular and also just stuff that's already out there.

So kind of pulling it back to the Salesforce world, what do you do with these little widgets? They are meant to be small enhancements to screens in Salesforce. From there, sort of wearing the Salesforce hat in their world, you have a record that represents an account or a contact or some other data, and you’re going to add another section to that record or replace a record with your on page. And their obsessions are security, declarative behavior and metadata, right? And it's pretty easy to sort of summarize those three.

And so as a developer, I can write a custom UI module. I can write a lightning web component with some logic to it may be backed by the database, may be interacts with the database in some way. And that little widget, I can make it surfaced for my admins that I work with. So the people that are not writing code, they’re a little more configurators and focused on sort of the

business flows. I can say, “Okay, here's a weather widget where it will show you whether it's raining or sunny or whatever at the location of the person whose record they’re looking at.” And so I can create the weather widget, and then in their tools, in the declarative tools where they go and sort of drag-and-drop and configure the UI, they can see both the standard widgets that they can use to construct that UI and custom widgets that people have created for them locally, but also installed. So Salesforce has an app store, much like a phone has an app store, and people can install complex business apps like the one that my company does, or they can install simple things like a weather widget for a contact. And then that admin can see those widgets from their screen and also configure them. So you can actually sort of surface configuration metadata for your widget and say, “Okay, I have colors. You can pick from these three colors. Here's the checkbox or a pick list or some kind.” So pretty robust little sort of widget system.

Where Salesforce kind of struggle, and this is sort of a historical Achilles' heel for Salesforce, is they were trying to build really complex things, standalone. And when I say that I still mean on the stack, but really not meant to enhance an existing screen, but be its own screen or a series of screens, like really your own single-page application. Some of the more abstract concepts, you want to use and knit things together, like session-based caches that you can invalidate sort of as you need to. Or some of the more sophisticated certain navigation behavior.

So, for example, Salesforce really controls the URL. They use the URL for a lot of their own stuff. And you can't really manipulate the URL from your components. So if you’re trying to, let's say, deep link to some state in your component, it can be really troublesome to do so because Salesforce expects the URL to look in it and appear a certain way so they can process it. So using it sort of at-scale for robust and complex applications is something that's challenging. And the community is working hard to sort of come up with patterns to make that possible.

So I said I wanted to go back to layers. Let me go back there real quick. One of the things, and this again, it’s just me speculating. I’ve been in the ecosystem a long time, about a decade. Salesforce has organically grown from what it was originally to what it is today, and today it is a robust, complex, sophisticated, general purpose platform to build stuff. And one of the things that’s sort of is interesting, is there database layer is incredibly sophisticated. All kinds of validations and logics you can tie to. When things get saved, what happens? All that behavior.

The UI layer is incredibly sophisticated. We just heard about lightning web components, and Aura, and Visualforce and all these different things that they've done. In the middle, it's always been a little bit lackluster. There is great sort of business logic tied to the database behavior, right? So if you want to react to records changing, if you want to interact records and mutate them or reject them as they are saved, there's a phenomenal support for sort of database-triggered behaviors. But if you want to do something every 10 seconds or you want to have some sort of daemon that runs and does logic or reacts to things that aren't strictly sort of database triggers, maybe time-based. It's really tricky to do in Salesforce. They don't really have a traditional sort of business logic layer where you can right sort of general-purpose stuff. And there's no way to import libraries.

So I worked with a company a few years ago that was trying to do natural language processing with Salesforce, and there are tons of great NLP libraries that exist out there, open source, that this company couldn't use CoreStack, because you can't import that sort of stuff, right? If it's in Apex, sure, but you can't import libraries from any other programming language. And so they were pretty stuck. And Salesforce bought Heroku some years ago now. And at the time I kind of thought, "Okay, maybe Heroku will become the middle." So like a sandwich and you got the top layer and the bottom layer, but you're the middle. And I thought Heroku might fill that gap. But really, it kind of hasn't. And the gap still exists. If I had one complain about sort of the holistic structure of Salesfoce was as a platform, is that middle layer is still pretty weak.

They've done a few different things to improve it. One, Heroku has helped, but they're really not as integrated with the CoreStack as I would like. But more recently, Salesforce is the focus on more tooling in that middle layer. There's a really great feature that they're working on called Salesforce Functions, and Salesforce Functions are essentially sort of like a Lambda with Amazon where it's a little, like a micro function that you're going to run and the compute is not yours to manage. So you essentially define the function and you make it part of your org or a part of your code setting. You say, "Okay, I want you to run this function asynchronously when these things happen," and Salesforce is going to manage some of their own computer. I assume that they're going to use Heroku to do this, but essentially their own version of Heroku that you don't have to manage yourself. And so you rewrite some sort of function or a complex behavior and it actually gets run off the CoreStack and the results of it are brought back in.

But as always, they've done a really good job from what I've seen with their early sort of versions they've' demoed of creating that metadata awareness so that your session is carried even though it's sort of technically off platform, because it's managed by Salesforce your session carries over. It's authenticate already, and you have access to other records in that same org, etc. etc.

So that sort of effort, something like Salesforce functions. That's not the silver bullet that's going to solve this problem, but I think it represents an effort by Salesforce to really address some of the challenges of the more complex behaviors people would need to do and want to do with their CoreStack and to continue to embrace this idea that they want to be sort of a fully featured general application building platform.

[00:32:32] JM: Could we zoom out and review that CoreStack again? So you've talked with the CoreStack several times. Just go through it once more.

[00:32:39] CL: So Salesforce, when they first kind of got started built a set of technology on top of an Oracle database, right? So a lot of different layers of business logic, all these stuff I've been talking about, Apex triggers, and Apex programming, classes, workflows, process builders, flows, all that exists in what I call the CoreStack. Salesforce changes their name for it all the time. It was Sales Cloud for a while, lightning platform. The name is a moving target, but this is a core set of technology that is most of what people think of when they think of Salesforce, right? The security model we talked about, that's CoreStack. So Salesforce has CoreStack. And then overtime they've acquired a ton of companies, right? So they acquired exact target and turned that into marketing cloud. They acquired MuleSoft and made integration cloud. They acquired – I forgot the name of the company, but the company became Commerce Cloud. So they've added all these pieces, and they bought Heroku. They keep acquiring other technology stacks.

And from a marketing perspective, the appearance is that it's all one platform, but that's really not true, right? There's the original sort of the core platform that they've built and have continued to develop over time, and then all these additional platforms they purchase that are possibly completely different technology stacks. And so the interoperability between these different sort of silos is a big part of the complexity if you're doing very sophisticated architecture on the

platform, is understanding sort of what lives in each silo and what sort of capacities exist for interacting between the two of them.

So the vast majority of people work almost exclusively CoreStack, that sales cloud, that service cloud, that's the ability to install programs from the app store. That's all sort of in one environment. Same database, same technology stack above it. And things that are different, things like Heroku. If you decide to use Heroku to do some work, you have to decide how you're going to get things back and forth. So Heroku is a different tech stack. That's a different database. You're probably running Postgres or something. And if you want your Heroku to behave alongside your CoreStack, you have to figure out, "Well, are we going to replicate data? What's the timing on that?" Now you are back in that world where you're writing coding you have to authenticate with your database and you have to make sure that it's there. And then if you're not connected for some reason, you have to have a fallback plan. So it reintroduces all this sort of craft and boilerplate that we are able to discard, because everything was sort of neatly coupled and tied together.

So when we talk about Salesforce at an architectural level, I think is really important to understand sort of these different parts of the overall platform that's owned by one business entity, but actually is discreetly different pieces of technology and how do they work together. And historically, I think Salesforce has not done an excellent job at integrating some of their acquisitions. Things like commerce cloud are still difficult interact with from CoreStack. It's just sort of confusingly so.

For a long time, marketing cloud and part of that were hard to interact with. Some of the stuff has been cleaned up, some of it hasn't. So that's sort of a sense of things. And so most the time when you're reading documentation or hearing people talk about Salesforce, it's CoreStack.

[00:35:43] JM: Tell me a little more about how you got so closely involved in the Salesforce ecosystem.

[00:35:48] CL: So I started doing Salesforce by accident. I think a lot of people start careers that way. So I was in college and I was doing Apex development for beer money. Someone asked me if I could write and Apex trigger. I said, "Sure. I know Java. I can figure this out." And I

started doing it more and more. And at the time, I was really focused on other things. When I first got out of school, I went and participated in an ice and water vending machine company. We've built this crazy piece of hardware, unique in the world. We had a patent on centripetal force used to dispense ice. So we built a telemetry system around the world to track these machines and do a complex UI. It was really a lot of fun. But as I said earlier in the conversation, I was a purist, and I believed in sort of the clean design principles. I really am one of those sort of crazy people that thinks code can be beautiful or elegant and kind of attractive, "Wow! That's great code."

And so sort of the thinking behind technology is really important to me, and that honestly is what attracted me originally to the Salesforce platform. The design, the idea that metadata is the king and everything stems from metadata. And some of the decisions they made about how things fit together. How the security model works. How the different declarative and programmatic layers work. They just got really, really right. And there're so many things that are annoying or wrong or I wish I could change about the stack, but the key design principles behind it are, in my opinion, completely correct. And I really haven't seen that replicated many places. A lot of systems you work with out there just don't quite get that piece right. And if you don't get that piece right, you're dead forever. You never recover from that.

And so I think that all this sort of theoretical ideas I had about patterns and design thinking I saw distilled in the product. And we talked about that a little earlier in the conversation about sort of the metadata and the ubiquitous language, Eric Evans and domain-driven design. And so that sort of what hooked me back in the day. And this is 10+ years ago. And then I've stuck around since, because in addition to really good sort of core principles for how they design their stack, they are incredibly sophisticated with their technology. They keep adding all kinds of very useful things. And I've gotten really lazy. So back in the day, you would construct a piece of hardware that you would run all your stuff on, right? And then you're like, "Oh, no. We're going to use data centers. And they're going to run the hardware, but I'm still going to install my own CentOS or my own Linux distro and maintain security patches." They're like, "Oh! No. No. No. I don't want to maintain security patches anymore. I'm going to let someone else manage the OS, and I'm just going to run the app layer. I'm just going to install my own sort of database on top and my other stuff, but they'll run the OS." And I've gotten so lazy now, I don't want to do anything with hardware, even like the bottom two-thirds layers was software. I don't want to run the OS. I don't

want to run the app layer. I don't want to run the database. I just want to write code and design systems that are at the shortest path to value for my customer and the people I work with as possible, right?

And I am really – This is one of the thing that Salesforce really got right, is they have done such a good job at abstracting and sort of removing as much of the technology stack as they could. So you really like kind of, day one, minute one, starting a project. You are already kind of working in the abstract world with ubiquitous language kind of thinking and doing data models and logic flows with people that already sort of match the way they perceive their data and their business and how it works. And you can really skip a lot of stuff. Now, is it not technical? No. It's incredibly technical still, because you have to understand all the nuances of what Salesforce did to with their stack. But you're not solely responsible for all that boilerplate over and over and over. So I really don't miss the boilerplate at all, and that's one of the things that I think has really hooked me about the platform and keeps me here.

[00:39:47] JM: Your company is called Valence. Explain what Valence does.

[00:39:52] CL: So Valence is an app in the app store, I mentioned. It's something called the app exchange, it's a way for a business to purchase an app and add it on to their org to do different things. And so I was a consultant for a long time helping people to do complex projects. I talked about those a little bit previously in our conversation. And one of the things that we kept being asked to do over and over was build integrations for people and very sophisticated ones, different types of systems, complex behaviors, may be bidirectional, bidirectional real-time with a reconciliation step, deltas, all kinds of complex stuff. But integrations are mostly the same.

Every integration is unique and it has to be kind of treated as its own special thing, but most of what you want to do is the same. You want delta sync. You want good air handling. You want a lot of diagnostics available to you. You want some sort of schema where behavior. And we found that with Salesforce, because Salesforce is so data-driven, right? It really lives and dies based on the data quality. Everyone that uses Salesforce wants to move data in and out of the org. And there are a lot of tools out there that help mechanically move data. A lot of middleware platforms is a very saturated market, but we couldn't find kind of the right fit, the Goldilocks tool for us that would make Salesforce specific integrations easier.

So Valence, that's sort of the preamble, is in integration middleware applications. So you install into a Salesforce org and it helps people that use Salesforce move data in and out. It's like a Jitterbit, or an Informatica, or a MuleSoft, and it's just kind of traditional middleware, right? It's got ETL type capacities, but it's run natively in Salesforce. So it's CoreStack, right? When it runs, it runs on that CoreStack. There is no other server involved at all. And that was really an unusual idea. And this ties back to our earlier conversation about the middle being a little bit soft in the Salesforce tech stack. It is really difficult to write complex code or declarative systems on the Salesforce course stack that aren't just tied to certain database transactions.

And so we built an entire middleware engine on CoreStack, and the compelling argument there, the reason that that justifies itself is because we wanted to understand Salesforce better than anyone else. This idea of Salesforce being so metadata-driven and so sort of sophisticated and tied to the org itself. We wanted to surface that with the middleware tool so that people could really just construct declaratively the connections they wanted to move data back and forth. They could design the formulas and logics for why things move or how things are transformed. But all that would live inside of that single Salesforce org. It wouldn't be a third-party stack somewhere that you're routing all your data through. So that was genesis of Valence, and that's what I do these days.

I really don't consult anymore or do as many big projects. I focus on my business. And that's been an interesting shift for me as an entrepreneur, is starting sort of a product-based business. Previously, I had been in a services-based business for so long. But Valence has been a really interesting project. We've been quite successful with it. The markets really responded to Valence, and it's been interesting to see people agree. So honestly, like many businesses, like a startup, we started it for us in some ways, right? We said, "Okay, this is something that irritates us. If it irritates us, it probably irritates other people like us. Can we build a tool that solves our problem that we've been having and then also might help other people?" And so that's been our focus, and it's really resonated well. Nothing else is quite like it. It's unusual to have a fully native solution like this for Salesforce.

[00:43:10] JM: As you begin to close off, what other kinds of topics would you like to explore around Salesforce?

[00:43:16] CL: I think we would be remiss if we didn't talk about governor limits. So because Salesforce is a multitenant environment and you wore code, your declarative stuff, your database sits alongside other customers in the same sort of hardware stack, same pod, you have to sort of respect your neighbors just like if you lived in an apartment building.

And so Salesforce has this thing called governor limits. There are all kinds of different ways they get applied, but essentially the long and short of it is that they exist to support the ability for everyone to sort of run on the same resources. And generally this isn't a big problem. It's not like your neighbor is doing some crazy computer job and your app slows down. Like as a shard or an instance in Salesforce, you don't notice any sort of effects from your neighbors, because these limits exists and have been around forever, and Salesforce does a really good job of enforcing them. So I don't want to give the impression that you are worried about people's impact on you. Please, you're really not, but Salesforce does have a lot of rules around what you can and can't do. And you are sort of compelled to understand these and respect them as you design and construct your solutions to things.

So there're some that are transaction-based, right? In a single execution context, which Salesforce calls a transaction. You can only do so many things. Let's say you can only do 100 discrete calls to the database to mutate records. So you're like, "Okay. Well, maybe I can only mutate a hundred records." That's not true. Salesforce has bulk versions of these mutations. So you can mutate up to 10,000 records in a single transaction, but you can't do it across more than 100 discrete calls. You want to make sure that you're doing things like bulkification where you're thinking about, "Okay, I'm going to collect all my changes together maybe even using a sophisticated pattern like unit of work." And then I'm going to sort of persist these changes as a set to Salesforce."

So it really forces you to think in a sort of cleaner design principles. The governor limits aren't arbitrary. They are based on sort of natural SKU of how people use a system like this. So they in some ways force you to be a better architect or a better developer because you have to think about what's actually happening under the hood a little bit, right? You don't have to understand how the database model inherently saves records. Just as an aside, it's actually a super huge table where all the data is in the same table and Salesforce is doing all kinds of crazy joins

under the hood, but they optimize the database. They optimize query performance. You just have to understand how that impacts you at the top of the stack through things like governor limits.

So part of, I would say being good, at doing Salesforce is understanding the restrictions. And I'm going to say this, and hopefully this resonates for some people. I think actually restrictions can really breed a lot of creativity in all kinds of systems, really, not just software, all kinds of places. You only have so many colors and you're painting or something, right? It kind of forces you to think outside the box and be more creative.

So I found that working alongside the limits has actually made me much more effective as an architecture or as a developer, because I sort of understand what's available to me and try to come up with a creative solution. So I think that that's sort of characterization actually fits the business. So Velance, the integration system. Because it's CoreStack, we are completely at the mercy of the governor limits, right? We don't run off platform. So we have to respect all these limits. We have to stick within them. So I would save half of the work that we've done, half of the thousands of hours we put in on Valence is just making the governor limits seem transparent and like they don't exist.

I'll give you that that number again, either 10,000 records you can write in a single transaction in Salesforce. You try to write 10,001 records in one transaction, you're stuck, you can't. It doesn't mean you can never save more than 10,000 records at once. It just means you need to get creative. So for example, if you give Valence 15,000 records or 50,000 records in one transaction, Valence will write all of them to the database. Well, how does it do that? Well, because of governor limits, we have to chop it up. So we actually kind of batch that dataset under the hood transparently, and we take the first 10,000, we write them, and then we serialize and save the remaining set and start another execution of transaction after. So you can chain these together in Salesforce using something called a Queueable, and you can actually sort of serialize stuff and then pick it up on the next execution context. There are all kinds of things that you're essentially forced to consider for the scaling of the work that you do because of the multitenant environment, and that's sort of governor limits in a nutshell. There's obviously a huge amount of nuance to that. We could probably spend 30 minutes just talking about governor limits, but that is a big part of approaching the platform, is understand those exist and

just being aware that you'll have to come up with a game plan for how you kind of work your way around them.

[00:47:48] JM: Well, that seems like a good place to end the conversation. Thanks for coming on the show, Chuck.

[00:47:53] CL: I really enjoyed it, Jeff.

[END]