EPISODE 1255

[INTRODUCTION]

[0:00:00.3] JM: Microservice architecture has become very common over the past few years, because of the availability of containers and container orchestrators, like Kubernetes. While containers are positive for scaling apps and making them more available, they've also introduced hurdles, like persisting data and state and container restarts, or pod failures.

Development teams put significant work into designing applications that take these hurdles into account, because without precautions, you can lose valuable data or crash your app. The company, Temporal, provides tools for both building complex microservices, as well as apps that use microservices. They use two primary function types; workflow and activity.

Workflow functions persist all local variables and threads, so if the server the app runs on crashes, it's picked up on a different server where it left off down the line. Activity functions automatically initiate retry logic if the service the function invokes fails for something like its server being down. Temporal provides visibility into end-to-end workflows that can span multiple services.

In this episode, we talked to Ryland Goldstein, Head of Product at Temporal. Previously, he was the Lead Project Manager at Reshuffle and Software Engineer Lead at Parallel Machines. We discussed the challenge of managing state and microservices, orchestrating microservices and how Temporal simplifies this process for development teams.

A few announcements before we get started. One, if you like Clubhouse, subscribe to the club for Software Daily on Clubhouse. It's just Software daily. We'll be doing some interesting Clubhouse sessions within the next few weeks. Two, if you are looking for a job, we are hiring a variety of roles. We're looking for a social media manager. We're looking for a graphic designer, and we're looking for writers. If you are interested in contributing content to Software Engineering Daily, or even if you're a podcaster and you're curious about how to get involved, we are looking for people with interesting backgrounds, who can contribute to Software Engineering Daily.

Again, mostly we're looking for social media help and design help. If you're a writer, or a podcaster, we'd also love to hear from you. You can send me an email with your resume, jeff@softwareengineeringdaily.com. That's jeff@softwareengineeringdaily.com.

[00:02:17] JM: Ryland. Welcome to the show.

[00:02:18] RG: Thank you. Happy to be here.

[00:02:20] JM: We've done a show about Temporal before, and workflow engines in general. I think this is a pretty deep subject. I'd like to start off by just exploring the distributed systems related problems that a typical infrastructure team might encounter that are still not solved by all these nice tools that we have today, like AWS and Kubernetes and so on. What are the outstanding problems?

[00:02:50] RG: Yeah, it's a great question. From my point of view, it's actually the same problems that people have been having, even before distributed systems were in Vogue, and they were the way to build applications. I think, the same problems around transactionality and atomicity, basically, guaranteeing that things like, transferring money works out well. Those were problems that existed far before most systems were distributed and there were all these web scale companies.

I think what ended up happening is that those problems, they lingered. Those are still challenges that companies that are really important, like banks and other financial institutions, even just e-commerce, they still have to solve those problems. Now, it's under the context of having to do it in a distributed environment. It's essentially having to solve the same problems that were already challenging, but now the pieces that you're actually building your application on top of and the things that you're relying on are shifting under you, because there are these distributed systems with all these edge cases.

[00:03:45] JM: Why is there a solution to this plethora of edge cases? How is that possible? If we have all these distributed systems edge cases, it seems like, they would have to be solved through piecemeal solutions.

[00:03:59] RG: Yeah. It's a really good question, because it's the mindset and mentality that a lot of the people that we even work with have when they start looking at Temporal and start evaluating Temporal. I can't tell you the number of times I've talked to a candidate, or a potential user who has pretty much told me that within one of their companies in their past experiences, they've built something that's almost identical, or very similar to Temporal.

When you actually get down into it, what you realize is that while there are a lot of similarities in terms of the goals and the needs are the same and the surface area of what they build looks similar, what they built is a very vertical version of what Temporal is. Their solution is very tailored to these specific set of problems and challenges that are relevant in their domain, but they don't go after all the generic edge cases of a distributed system.

What ends up happening in those cases is that maybe for that initial use case where they build this Temporal lite, things worked out really well. When they need to start bringing in more scope, which is inevitable for basically any business need, and the boss comes and says, “Hey, you need to add this new feature. We need to support this new customer.” Well, now you're adding these new features on top of this Temporal lite that you were initially scoping it for. You start having a lot of debt, and you start having these maintenance issues and things become very, very challenging.

Eventually, you reach a point where you've pretty much built a poor man's generic Temporal. The problem is that you didn't take into account all of these requirements at the outset. You didn't take these requirements into account when you started building the thing. You have a very sub-optimal solution for this now generic problem. This is a position that a majority of our users find themselves in, who are coming from existing solutions. Even just a lot of people I talk to, or developing software in the industry.

[00:05:37] JM: Can you give me a specific example of a problem that a technology like Temporal can solve?

[00:05:47] RG: Yeah, absolutely. I mean, harkening back to that financial use case I alluded to, I think the most simple one is just the money transfer. If you want to have a process where money is taken from one account and put into a different account, and the money is stored in different banks, or accessed through different services, those services don't have any way of surfacing a transaction between each other.

The canonical problem you have is that you send a request to do the withdraw. Then, you want to do a deposit afterwards. If there's some failure that happens between those two steps, you now have the level of inconsistency in your application. Specifically, in the context of transferring money, this is a really big and very visible problem. Because people, they hate losing money. When you're actually dealing with money as your business, it's very easy to see when it's lost, as opposed to if it's something more virtual, or less tangible, sometimes it's harder to tie it back to the money and the finances, although it's almost always possible.

[00:06:40] JM: Can you give me a description of the Temporal API and how companies use it in general?

[00:06:48] RG: Yeah. The thing that's a bit interesting about Temporal is that for us, an API is a lot different than what you would have with a traditional software as a service, or any level of service. Most people, when they think about an API, their minds go to what postman does, where there's a set of endpoints that you call and the way that you integrate that API into your application is by making RPC calls, HTTP calls, whatever. You may come to these endpoints that are well known and that's your contract that you have with that service.

In Temporal, the approach is quite different. Because the majority of the time, users aren't just writing RPC calls that call our back-end. What actually happens is a much deeper integration into the application itself. We have these client-side libraries that are provided in a few popular languages and we support Go, Java, and we have a node SDK that's coming out right now, also PHP.

Essentially, what happens is that you have to take a different approach to building your application, in general, when you're using Temporal. Instead of just having a function, which has a bunch of routes and you have a HTTP server, with Temporal, you build your applications with building blocks, which we call workflows and activities.

All of the usage, generally, of Temporal's APIs are through these activity and workflow functions, which encapsulate your business logic. Actually, a lot of the usage of Temporal, that's actually the extent of it. If you're writing a very simple application and that you just need to orchestrate activities from your workflow, you probably don't need to use any of the other APIs of Temporal. We offer things, like a way to process events that are coming into that workflow and a way to read data from the workflow. For a very simple money transfer use case, you might only need to call an activity and that's the gist of your workflow.

That's the interesting thing about temporal is that API usage is, it means something a lot different. Now, obviously, we do have APIs to some degree, for things like visibility layer that we surface, which allows you to get some top-down insights into your running application. We do have APIs for that. In terms of the core path of usage of Temporal, it's not really traditionally API-driven. It's more that you write code. The way that we've integrated with the client, it allows us to introspect and know when events are being generated from that code and statefully persist them on our back-end.

[00:08:57] JM: What are some of the failure cases in that state kind of management?

[00:09:02] RG: Yeah. There's a lot of them, especially when you start talking about the generic problem, which again, is the core value that Temporal provides to users today. I think, there's obvious errors, like the machine that you're running your code on crashes. The Temporal service crashes, the downstream that you're depending on isn't available. There's this plethora of different situations and different scenarios, and even ones with the same components, if they happen in a different order, maybe the way that you want to handle them is different.

I think, the thing is that most failure cases that people care about are actually at the application level. Your downstream isn't down, or maybe the availability isn't as great as you would have expected. Now, you have to basically implement all these boilerplate logic and state management to ensure that you can manage the lifecycle of that downstream service being up or down in a given point in time.

People end up building the majority of state machines that we see our users build when they come from an existing solution, are all around these application with failures, where you're coordinating a bunch of different services; maybe ones that you own, maybe ones that you don't own. You can never really guarantee if this is the core problem of distributed systems, is that it's not you, it's not the thing that is currently running your application. Therefore, you can't control whether it's up or down in a given point in time.

Even if you're really confident that this downstream service, maybe S3, who has 10 million nines at this point, you're pretty confident that that service is going to be available. You probably still want to code within your application around the case where it's not available, because it only takes one time. Maybe all of a sudden, that sends your application haywire. Now, you're charging people money that you shouldn't be doing.

I think this is the core problem is that there's all these failure conditions, which are basically, domain specific to the way that you write your application. The framework and tooling that people rely on today, for the most part, doesn't really give you a way of generically handling those application failures. It's left to the developer to handle them on a case-by-case basis.

[00:10:52] JM: These kinds of problems were solved in different ways by companies that had to implement solutions. What are the hackneyed, subpar solutions that are used if people are not using Temporal? Is it just retries and all kinds of caching stuff? What do people do to try to get around these sorts of problems without some framework, or tool chain?

[00:11:18] RG: Absolutely. Yeah, as you said, I think the most predominant one, by far that we see are the ad hoc solutions. The one thing is that there's not a lot of people that come to us with a single system that they're then replacing with Temporal. In almost all cases, they're coming to us with a set of different dependencies that they've chained together to provide the set of guarantees, which they feel are very, very important for their application.

There's some very common patterns here. I think, the most common and the one that we often talk about are the state machine problems, which is again, you have two different services, which are maybe running on different databases. You can't have a formal transaction between them. You want to make sure that if one of these things execute successfully, and then there's a failure in between them, that you have a ability to have the similar behavior you'd get with a traditional transaction, where it either completely succeeds, or it completely fails. There's never this middle period in between.

What most people end up doing to solve those problems is they construct these implicit state machines. To do that, they rely on things like machinery, or Redis, or Celery, or whatever your solution is to broad, basically, these cues and these persistence layers, where they can push these intermediary events that represent the state between these transactions. That's just the infrastructure side of the solution. You need to have a database. You need to have a queue. You need to have all these things.

There's also the burden that puts on the application itself, which is that now when you write your code, you can't assume that that code is starting from the beginning of where may have at the top of your code, the first line, because you could be entering an instance of execution that represents something that previously failed in the past. Now, you have to burden your application logic with all of these controls in different edge cases where you say, “Okay. Well, if I'm starting with this state, then I don't need to do these steps.” You end up adding all this boilerplate and complexity to your application. The whole goal the entire time was just to have a piece of code that runs from beginning to end without issues.

[00:13:12] JM: I think, I have an understanding of what Temporal does for the user at this point. I'd like to talk a little bit more about the usage of Temporal, the company, versus the open source projects around it. Can you describe what the hosted service does, relative to the open source frameworks?

[00:13:37] RG: Yeah, absolutely. Today, the answer is not that complex. We brought up our cloud solution, I think, four or five months ago at this point. Our initial goal was that we had this huge group of users who weren't interested in running and managing their own Temporal setup internally. For those people, I think it's a basic, they don't want to run infrastructure thing. There's no hosted version of Temporal, other than the one that we provide and we started providing. There was a very primitive need, which is we don't want to run our own infrastructure.

I think, a lot of our users today, a lot of our customers, that's the category they fall in. Now obviously, the benefit, even in that case where you're differentiated just by the fact that you're

putting software on servers is that we're the ones who have the biggest mindshare of Temporal. The creators of the technology for the last 10 years. They're in the business. They're able to provide you support and assistance. That's the vision that they're going to be able to provide to the product, both the customer and the user level.
I think that's a big thing for our users and our customers today. What I think a lot of our customers are buying into is the future prospects; the things that we're going to be able to do with the cloud service and the cloud offering, which just aren't going to be possible in the open source. I think there's a lot of things just about the existing model, which we plan to improve and make even fundamentally better.

Right now, the system is known for being quite scalable, and very, very durable. I think we're just going to make that even more true. For example, if you really want to run at a large scale with Temporal today, you have to rely on the Cassandra database underneath. There's a lot of reasons for this, but the biggest one is that the other database solutions that we support with Temporal, they don't have the greatest horizontal scaling story. That's our bottleneck when we design things at the system level.

One of our biggest things is making sure that the experiences with other databases are also good, because right now, there's this huge barrier of entry for people who want to use the product at a scale that is significant, they have to use Cassandra. Right now, we're advertising. Our cloud is the solution to that problem. Regardless of what database we're using, we can guarantee the scalability and reliability, things at any scale. That way, you don't even have to worry about what database is being used under the hood.

Outside of the raw performance improvements and the baseline stuff that we want to do, there's also a really strong vision there, about the way that people are building applications today, about the way that people are collaborating with different teams on building those applications. Temporal, accidentally, is positioned in the central place to solve all those problems. We think that we can add a lot of value-ads on top of the existing offering, which we think is already pretty robust and feature complete, but we think we can add things like for example, analytics. We might even be able to tell you when you have your application running on Temporal, which parts of your application are slowing things down, or which parts of your application are performing as well as they could be?

The reason that we can do this is because there's this engineer in the company who likes to call it inversion of execution, right? We flipped the traditional execution model. Even though Temporal cloud isn't running your code, it's sending all these events, like your workers and your applications are sending these events, which represent the outline of how your code works, of your business logic.

If we implement the right things, we can actually introspect that and make pretty intelligent decisions and observations about the way that your application is flowing through the system. Even recommend things like, “Hey, there's a bottleneck in this part of your application. If you restructure things like this, it's going to have a huge improvement in performance.” That's without ever actually seeing the implementation of the code itself.

This is actually not theoretical. There's a customer that we work with, who basically had this happen. We were getting some alerts on our cloud. One of the engineers looked into them. They noticed them and they realized very immediately that it wasn't going to be a problem for our service itself. The cloud was not in danger in any way, but that the user was basically not running the application efficiently, and that there were these bottlenecks. From our point of view, it almost made us more money.

There wasn't really an economic motivation, but we felt was really our job as a developer company to go to this customer and basically tell them, “Hey, look. We looked at the events that are being sent from your workers and we really believe that there's a way that you can change things to make that better for you and better for your experience, better for your users, and also, probably save you some money.”

We were actually able to go and tell them exactly how to fix this problem that they had in their application, even though we had never seen their source code, just from the data that we're getting from the workers in terms of events.

[00:17:54] JM: Tell me a little bit about the product design process, as you are the head of product at the company, or maybe not the head of product. You work on the product. You are the head of product.

[00:18:04] RG: No, no. I'm head of product. It’s good.

[00:18:06] JM: You are head of product. Yeah. Tell me a little bit about the product design process and just the general engineering management side of things, like what does the product development process look like? How have you iterated over time?

[00:18:20] RG: Yeah. It's a very nice question, because it's definitely a non-traditional answer for Temporal compared to most companies. I think one of the biggest things that from the product side that I go up against Temporal is that the technology is sufficiently complex, that most of our users really don't have a deep understanding of how it works.

What that manifests as from the product point of view is that people don't often have really fundamental feedback about the way that the product works, or what features they're looking for. Obviously, we get a lot of high-level features, very surface level things, like you should support SAML, or you support OIDC, or any of these things. When it comes to changes to the way that the product fundamentally functions, there's very rarely feedback that comes from users directly.

We have to do a lot of work, basically, to go and understand what are the real needs of these users. Because in most cases, our users are experiencing actual problems, ones that if we change the product, we could definitely solve them. The issue is that they don't know how to communicate those. They don't even understand that they're having a problem, which Temporal could even solve.

I would say that user research and just understanding our users and developers is probably something that we spend more time doing as a product function than anything else. Even though that's a traditional part of the product process, I would say that the amount of time we spend doing it is definitely disproportionate, compared to the companies that I've been and at least the experiences I know of.

That's definitely a huge part of it. I think when it comes down to actually designing things, again, we have to take in the requirements and the needs of what we see our users doing. In terms of

how the actual API, so to speak, or design and the surface area of the thing, that's something that is very much left up to the product and engineering side of within Temporal itself. I think we use very traditional processes to some degree, in terms of we're agile and we try to do things where we have a real design before we start doing any work.
I think, what's different is that there's not as much of a validation phase where we can be like, "Oh, is this, we can go ask users, "Hey, is this going to work for you?" Because a lot of the times, the product level feature is something that's only internal to the distributed system. That's definitely a huge challenge that we have from the product side of things. I think the one thing that's very interesting about Temporal is that everyone is very technical at this stage.

I'm the head of product of the company. I'm definitely not one of the least technical people in Temporal. My background is almost completely doing development in distributed systems specifically. Even the rest of the product team that I manage, almost every single one of them is a proper engineer. That means that we have a lot of different processes in general, because pretty much everyone can code, pretty much everyone understands basic distributed systems theory and concurrency and parallelism. People are a lot more cross functional in that sense, where somebody from the product team can do a pretty good job of understanding what the engineering requirements will be based on something that a user asked for.

I think that we have a lot more collaborative and very, very, integrated process in that sense, where engineering often does a lot of product work and product does a lot of engineering work, which definitely leads to a very interesting environment. I think, as an engineer, it's a really great environment, honestly, because things feel very, very collaborative and mutual. Yeah. I think that's the gist of it.

I think with distributed systems, the biggest problem with the design is that there's all these implications about how things work. It's very easy for me to go to the engineering team and say, "Hey, the account model should be designed in this way. That's the ideal user experience." As somebody who has built a lot of distributed systems, I know what the implications of the structure of our accounts, for example, will have on the way that the system is actually built.

That's definitely also another huge consideration is that we have a pretty mature product and there's a lot of lines of code, a lot of, basically, investment in the way that things are built today.

You really have to make sure that you think through the way that your changes and the things that you want to do from the user standpoint are going to actually impact the guarantees and the performance of the system itself.

[00:22:06] JM: Are there any bugs that users have discovered, that maybe took you by surprise?

[00:22:13] RG: I'm racking my brain to try to remember a specific situation. I think, the one that I can remember that a user came up with was something around our tenancy model, that we didn't understand that there was an edge case that you can basically call a resource from one environment, what we call namespaces, to another environment.

At the time, we were in the very early stages of adding authentication and authorization into the system in general. It wasn't really that much of a security gap, because there wasn't really even security in the product yet. In a bigger picture sense, it was a very generic security vulnerability, because you could essentially access unauthorized resources. That's something that we basically had in thinking about the product from that point of view, for the lifetime up until we added those features.

It was something where we hadn't even thought about those things when designing the features originally. We were a bit surprised, because we hadn't even gone back and retroactively thought, “Okay, how did these changes affect these specific set of situations?” Again, it goes down to with a distributed system, there's all these edge cases, and Temporal tries to formalize those, and making sure that every single decision and product feature, like authorization and authentication propagates down to all of those different edge cases, is definitely a real challenge.

Then, I think, maybe just to add a unrequested one. Not a bug that wasn't necessarily discovered by user in the sense that they were looking through the code, but we did have an incident that we wrote about not too long ago, where we had a patch, essentially, that was put into the system that introduced a bug with Cassandra. The bug was introduced in our Golang code, because there's this basically, shadowing functionality in Go for variables. The shadowing led to the code being very, very difficult to reason about and understand. We actually missed this bug. There was an assignment. Essentially, the only time that this assignment could happen is if Cassandra itself was experiencing issues.

Basically, what happened is that we discovered that bug and we realized, “Okay, this is a very rare thing to happen. Not that many people right now are running on Cassandra. There isn't that much reason to be worried.” Essentially, that wasn't true. We made a big mistake there. We realized, because many months went by and we had already patched that thing, but we hadn't made a broadcast about it. We hadn't gone and told people, this is a big deal.

Then one of our users, a relatively large user was running Cassandra in production, and basically, was doing some upgrades to their Cassandra deployment. That triggered this edge case happening with the Go shadowing. It created this huge, huge incident and huge problem on our side, where we realized there is users who are still running on the old versions, we should have broadcast this better. We just never expected that somebody was going to run into that edge case, because we felt it was so rare. That really changed our thinking about how we communicate things to our users, how much we increase the severity of certain incidents based on their potential and not just based on how likely they are to happen. Just generally, our empathy for these situations and the way that they can impact people.

[00:25:08] JM: Do you think Temporal, or Cadence, Cadence being the open source workflow engine, do you think it can build the community that Kubernetes has built, or an AWS is built? Or do you think the addressable market is different, or smaller?

[00:25:28] RG: Yeah. I think in terms of size, there's a huge market for this technology. I think the market just grows every day by a huge, even maybe order of magnitude, because more and more companies are accidentally finding themselves in the position of being distributed systems companies. This is coming from two different directions, because nowadays, any new startup that's in the software space, they realize that they need to run at a global scale. They realize they need to be on the cloud.

From the beginning, those companies are basically distributed systems. With the older style of company, the old guard that have been around for a longer amount of time, for a long time, they were battling this, and they wanted to stay on-prem and they didn't want to be on the cloud and they wanted to keep this monolithic architecture. The forcing functions of nature and the environment are not allowing that to happen anymore. They're now being pushed to adopt the cloud, and start distributing things and using service-based architectures.

I think, both of those are essentially funnels into the potential market for technologies like Temporal and Cadence. I definitely don't see that changing anytime soon. I think, specifically the question about Kubernetes, I absolutely think in terms of the usage and the adoption and the impact, Temporal can be at the level of Kubernetes, I would say. No offense meant to Kubernetes. I think the one thing that Temporal might be able to do is be a technology that people love to use.

As somebody who's used Kubernetes once or twice, at least, I can tell you that I think Kubernetes is one of the most valuable technologies that's around today. I think that we'd be a lot less far along in terms of the way that companies use software and that services are provided if Kubernetes hadn't been created, and if people weren't using it so much. I think it's very rare that people are truly in love with Kubernetes and they feel this is something they want to be spending their time doing, and that it's really the differentiated value.

I think, there's obviously people who they're very passionate about Kubernetes. It's not a 100% true. I think, when you look at developers specifically, and not people who are in the more ops or infrastructure side of things. Usually, they see Kubernetes as a necessary evil, right? They don't see it as something where they're like, “Wow. I'm waking up every day, so I can go like an operator for Kubernetes, or something like that.”

I think with Temporal, the big differences is that the developers love the product. Developers love the way that it changes how they write their applications and it's not something where they feel like, but grudgingly, “Oh, we're going to use this, because otherwise, we won't be able to scale our business.” It's almost always the opposite, where they're like, “Our business is already having issues scaling. We need a solution there.” It seems like Temporal not only solves our problem, but also makes our lives 10 times easier.

I think that's something that even AWS has never really been able to get down, where they provide this incredibly broad catalog of services, and it pretty much solves any need that you have. If you need to run at scale and you need all these requirements, I mean, AWS is pretty much one of the only options today.

I think, there's definitely not trying to talk AWS down. It's done an amazing thing and solved a lot of fundamental problems for the software industry. Again, it's not one of those things where people are usually that passionate. Developers aren't dying to use AWS. In fact, if I had to say what the weakest part of AWS is, it's the developer experience, it’s user experience. It seems like, something that AWS has always struggled with. From their position, economically, they're always very customer motivated, but it's always to the point where they're need is satisfied, the basic need is satisfied.

AWS is not known for doing polish on things and really trying to do a ton of value added on the existing service. It's more about the breadth and can we get the majority of adoption? I think, that's one area where Temporal can definitely make a lot of distance between Kubernetes and AWS-like solutions, which are more focused on the practical, functional side of solving the problem. Whereas, Temporal is not only addressing that aspect of it, but also saying, “Okay, look. There's a better way of doing this in general. Life doesn't need to be as hard as it is right now.

[00:29:14] JM: I think, I remember asking Maxime about this when I interviewed him, but how do you see the difference between the kinds of state management problems faced by the DAG workflow engines, the airflow suite of stuff, airflow, or prefect, or DAGster, versus the kinds of microservice-based applications we're talking about here?

[00:29:40] RG: Yeah. I think, aside from the cases where people tried to use one technology incorrectly to solve a problem that they shouldn't be trying to solve with it, I think there are some very valid places where the DAG-based approach makes sense for building stuff. I think, if you try to take a generic service-based application and then you write it in code and have to generate a DAG, almost never is that going to be the optimal solution.

Mostly, because I think code is the most expressive way to solve these dynamic problems with services and communication. I think the area where solutions with DAG-based architectures really excel, is when you know the entire boundary of the problem upfront. A great example, that is if you're doing stuff with data processing and maybe even machine learning, there's often a very rigid set of steps that you have to take on a set of data to transform it into feature engineering and make sure that you've sanitized it and then run your basic linear regression.

All of those things are pretty much known at the outset, because it's very mathematical. It's very scholarly, for lack of a better way of putting it. I think, for those cases, it makes a lot of sense to have a DAG. Mostly, because one of the huge benefits of a DAG is that once you generate it in the initialization phase, you also have the chance to optimize it. Just as with the compiler, if you have access to all of the code beforehand, you can make some really, really generous optimizations, because you know the full scope of the problem.

I think the thing is, is that as the scope of the problem becomes less defined, and if there is more dynamic places for things behave differently, the value of optimizing that DAG go down considerably. The DAG becomes so complex, that you can't have a bounded way of optimizing it and constraining it, almost similar to a traveling salesman problem, where there's just too many different things, there's too many different ways, that it's very difficult to create a bounded solution to that problem.

I think, that's a very similar inflection point for when a DAG basis that makes sense, or it doesn't make sense, where if you have this very constrained problem, where you know you're going to need to do this thing and then that thing, and the order is very set in stone and there's not that much branching, having a DAG and optimizing it can give you a lot of bang for your buck.

The moment that you need to make any decisions within that DAG and start doing stuff based on the way the previous steps evaluated, then you're going to start running into some really, really fundamental issues. We even know of stories of people who came from solutions, like Airflow and they had an airflow application, or Airflow pipeline that would run for maybe 10, or 15 minutes, but it would actually take more than double that just to compile the DAG to run the thing in the first place.

There's definitely a place where it just becomes more hassle than it's worth, and it doesn't provide you all that much value. Then Temporal, in general, is much simpler to write with. If you can afford to use it and it's not going to hurt your performance, it's absolutely a better solution, in my mind.

[00:32:16] JM: With that in mind, can you give me a little bit more about the long-term vision of the Temporal project, or the cadence project? What's the scope of the kinds of workflows that you expect to be able to take on? Or are there kinds of workflows that you think Cadence maybe cannot take on today, that in the future would be a good fit for?

[00:32:40] RG: Yeah. I can start with what we can do today, and then I'll enter the vision portion. For the what we can do today, for the most part, the big barrier for Temporal applications, and the one thing that is really important for people is that it usually makes sense to use Temporal of durability and reliability is important. If you're sending a bunch of – if you're sending an email blast to 10 million people and you don't really care whether all the emails get delivered, say that you only need 90%, or something, even 99% of the emails to be delivered. You're not going to get a lot of value from Temporal today.

There's an argument to be made, that there's a lot of value in Temporal around the orchestration and the expressivity of the code itself, and the way that you build the application, but there's a lot of basically, work that's been done specifically just around the durability side of things. If you don't care about durability, you're basically adding overhead to your application that you wouldn't otherwise need, like basically, having to go to the database after everything that you do.

I think in general, that's the best way to think about which use cases today are not the best for Temporal or Cadence, is that if you can't afford to go to the database after doing anything stateful, then it probably doesn't make sense for you to be using Temporal. A good example there is if something super latency-sensitive. You were doing VR, AR and you need sub-millisecond, maybe even millisecond latency, you're going to have some real challenges doing that with Temporal. It's not a good fit.

Mostly, because again, anything you do that stateful, you have a minimum round trip to the database and back, which is let's say, even 10, 20 milliseconds. Very quickly, if you have a workflow that has a few activities, you're talking like an order of magnitude of 100 to 500 milliseconds to run that workflow. For relatively short-lived workflows that are seconds, or even maybe a hundreds of milliseconds to forever, it's a great fit. If you need those really, really fast workflows that are very performance sensitive, that's probably not a good fit for us.

Now, there are definitely cases where people want the latency sensitive workflows and they also want reliability and durability. In our experience, there's not that many, and most of the time, you don't really care about those things as much. In general, if you don't care about the statefulness of your process, then it's probably not worth using Temporal today. In terms of our vision, I think without even going to the stateless compute side of things, I think our vision is that 10 years from now, anyone who's going to build an application, and I think in 10 years, pretty much every application is going to be connected or AKA distributed, we think that they should be doing that with Temporal.

Even more fundamentally than that, I think Temporal isn't just a arbitrary set of features that are tied together at a point in time. I think, Temporal is really converging on what the set of cases in a distributed system, or that can go wrong. What are all the different building blocks of a distributed system that you need to have access to and be aware of to build all of these different combinations and different manifestations of a distributed application?

I think, I can easily see Temporal being the framework that essentially, all applications are written on in the future. If somebody is coming out of a boot camp, I would love in 10 years, if they use Temporal not just because we've done a really good job of selling things and going on the right podcast, but because it's truly the best way of developing that application. It's the process that requires the least overhead and the least thinking about problems that you don't want to solve, and just allows you to focus on the thing that's actually differentiated for your business, or for your use case, or even just for your project.

I think, that's the overall vision. I think, one of the things is right now, we solve a very specific set of distributed systems problems. I think there's definitely ones for example, if you need a big storage engine and big data processing and all of that, we don't solve those problems today. We don’t have solutions for those. I would love to figure out ways to solve those problems.

I think, the one thing is that we don't want to do it the way that has been traditionally done, where you say, “Oh, you have a lot of storage needs, let's go build a really big storage back-end.” I think, requiring people to basically use it in the most naïve, direct way is unfair. It's not really empathetic, because people don't want to go and have to learn a new way to store things, or a new storage system. They want things to be stored and they want to have a solution that solves all their problems and is reliable and scalable.

If they didn't have to think about it, I mean, if you take a company, like for example, Coinbase is one of our users, they don't have any value in going and building a distributed storage engine, or understanding and learning a new storage engine. If their only goal is to store things for their users or their customers, that's all undifferentiated. Any work that they do that isn't just writing the business logic to achieve that goal is basically a waste of time for them.
I think that if I sum up the Temporal vision in one way, it's minimizing the amount of time that you do things that are beneficial to your business, and minimizing the amount of time developers spend writing code, which isn't directly valuable to the thing that they're trying to do.

[00:37:18] JM: As the head of product, can you take me inside an average day for you? What is your engagement with customers like? How are you interfacing between the customers and the engineering teams that you work with? Just, how do those kinds of customer demands get translated into product decisions?

[00:37:38] RG: This is going to be an interesting answer. Because at this point, I'm filling a few different roles within the company. I'm in charge of the product side of things. Also, as of right now, I'm the one who's responsible for the sales processes, like getting the initial customers, the one who does most of the interaction with users. I'm starting to build out this team and we have some already amazing people on it. In terms of the outbound work, myself and even the founders are the ones who are doing the most of the heavy lifting there.

I spend a lot of my time even playing a solution architectural. I think, that's one of the really great values I have, because I do have this developer background. I've spent a lot of time writing distributed systems. A lot of the times, our customer engagement starts as me just helping a user with one of their problems, who's running on the open source version of Temporal. Eventually, I realized, “Hey, look. This user, this company, they don't want to be running Temporal themselves. There's no value.” Just in the same way that there's no value in building all this distributed systems, boilerplate and distributed systems like infrastructure to solve your business use case, there's no point in running that thing if it doesn't provide you differentiated value either.

I have a lot of customers that I just get organically from the point of view of me working with them as users, sometimes we become friends, and then it becomes clear, and it just is a very easy conversation to have and say, “Hey, look. We have a hosted solution. You already trust us to write the technology, you could easily trust us to run this thing on the cloud as well.” It's a pretty nice process in that sense.

Obviously, it's not scalable. I'm a single person. We're actively trying to basically, create this structure within the company, so we can have these formal sales processes, where we go out and get users inorganically and not just the ones who are already within our ecosystem. Going back to the question about my day-to-day, I actually have a very diverse day-to-day. I spent a lot of time doing product stuff. Right now, for example, I'm doing a lot of product work around the new features we want to build in our cloud offering, a lot of this stuff around how the control plane at the cloud offering is going to be designed and how that will be surfaced to customers.

I do a lot of stuff around the more outbound solution architecture work, where I'm meeting with companies. We have a lot of huge companies, Fortune 100 companies who want to switch to Temporal. Oftentimes, they have a global distributed system already that's running in production, and they need to migrate that thing.

It's not just a case of well, it's a Greenfield project, go and start writing with Temporal. It's like, we have to have a real practical plan and strategy and timeline of how do you piecemeal this thing away, and eventually, have a fully temporal solution instead of what you're using already? That's the solution architecture piece.

Then also, there's a developer relations side and content and marketing. I feel it's very important for us to really start building our messaging and really start building our place within the ecosystem, even though we don't know how to place ourselves necessarily, like we think we're a fundamentally new category of software. It's really important that we at least are educating people and we really start getting awareness out about what you can do with Temporal, and what are the things that you shouldn't be doing with Temporal.

There's a lot of time spent on the relations side and the content and the marketing. Then also, more of from the engineering side. I try to make sure that I stay involved with those processes. I understand what we're doing with our control plane efforts and if we have changes that we want to make to our architecture that I'm aware of, that side of things as well. I would say, I play a pretty holistic role in that sense right now, just tying things across the board together, so we have a well-functioning machine.

[00:40:56] JM: Are there any notable changes that you've seen in software architecture, from your vantage point as managing product? Just trends in how developers are using new tools, whether it's serverless, or Google Cloud run, or whatever the open source version of that is, there's – I can't remember what it is. Knative maybe? I don't remember what it is. I just love to hear if you've seen any changes in architectural patterns.

[00:41:28] RG: I have to think about it for a sec. Obviously, the biggest one is Temporal itself, that people are embracing this formalized way to represent their systems. In terms of other things, I was on the ground for the serverless movement. My last company that I helped start, we were building a competitor to AWS lambda. I was there from day one. Initially, what is actually really interesting is that most serious developers, they actually didn't care about things like serverless and lambda. That's actually a problem we ran into on my last business, is that people thought it was very interesting conceptually, but when it came down to practically rebuilding their application with it, they're very resistant.

I think, there's the obvious one, which is it's a new model and people don't fully understand it, but there also just wasn't the belief that it could fundamentally solve the problems that people were experiencing. I think one thing I've seen is that over the last four or five years, developers have been buying more and more in this serverless thing. I think, part of it is that a lot of developers started having to do ops, and they're like, “Yeah. Well, that's not as fun as I expected it to be.”

Now, they're like, “Well, maybe there's a way that I don't have to do ops.” They're getting more interested in the serverless solutions, but I still don't think that anyone has truly delivered on the serverless dream in the way that most people expected when lambda was released, like I expected, at least when lambda was released. Interestingly enough, I think Temporal is probably one of the closest in that vein to solving the real serverless problem, because we're not providing an infrastructure specific way of solving it. We're saying like, “Here's this generic open source technology. You don't have to develop things with the understanding of the specific infrastructure primitives. You also can run it yourself.”

I think, like serverless, definitely, I've seen more and more interest from it from the ideological side. I don't see that many more people using it, but I feel a lot more people wanting to do something that serverless would represent, and solving those problems for them. I think, the other big thing is that the growth and the adoption of things, like Netlify and Vercel, definitely signals really important change in the way that developers are thinking and the way that companies are operating, which is that before, up until maybe even three or four years ago, developers were very much a means to an end.

It didn't really matter whether your developers were that happy. As long as they were happy enough, that they weren't writing bad software and they were building the system that you wanted them to build, you were good.

I think, the thing is that you can see how that manifested in the way that products are designed. Again, going back to AWS, it's not meant to be the most friendly thing in the world. It is not a great developer experience. I think, even AWS would probably admit that. I think, what I've seen, this big trend that I've been seeing is that the developer experience is something that has a financial amount associated with it now. People are actually taking real interest in that.

That's why you're seeing companies like Netlify and Vercel really gaining traction and people becoming very excited about them, because they're actually products which cater to developers. They're not a product, which developers have to use and are clunky and not so enjoyable. It's a product, which is first and foremost, made to improve the developer experience. I think that the big trend I've seen is that the world and the economy, or whatever it is, the powers that be, they really value developers now.

They really think that developers aren't just a way to accomplish the goal of building software for your business, but they're also an integral part of the business itself functioning well. Making them happy is a real way of making sure that your business is going to stay healthy and stay successful. I think, that's an awesome trend. I'm super-duper excited about it, because I think it opens up a lot of opportunities for people to bring in products, like Temporal into the ecosystem, and really differentiate in an area in a vertical, which has not been really valued up until now.

I think, that's going to be very empowering for developers and just users in general, because it's no longer just about the core, raw requirements and baseline things needed to have something up and running. It's also about how enjoyable is it to build that and maintain it and scale it?

[00:45:12] JM: Cool. Lastly, this is a pretty far-flung question, so you may not have an answer for it. Do you have any perspective on whether Temporal is, or Cadence I should say, is useful for solving any kinds of distributed systems problems that can emerge from crypto applications?
[00:45:36] RG: Yeah. There's a very funny joke that Shawn, who's our head of developer experience, he likes to say that it's inevitable at some point, there'll be a Temporal coin. I think, there's actually a lot of alignment and a lot of ideological similarities between them the blockchain and Temporal.

I think, the most obvious one is that there's this level of guarantees of the way that things have executed and an audibility of those executions. In Temporal, we have this notion, or concept of an event history, which is the list of all the events that have been generated from your running workflow, or your application, during its lifecycle that made changes to the state.

I think, from again, the ideological perspective, blockchain is very, very similar to that, where you want to have this ledger of transactions where you can prove at any point in time, what were the order of operations? How did they happen? You go back and make sure that you can trace through that and have a guarantee that things went the way that you expected them to go.

I think, just from that point alone, there's a lot of alignment. I think, in the sense of how can you leverage Temporal to make cryptocurrency, or blockchain solutions even better, you've already seen some of that. Again, we have a case study from Coinbase. They're obviously using Temporal or Cadence to pretty much orchestrate their core cryptocurrency transaction load.

I think that's a great example, where Temporal is very synergistic. It's not just there's an ideological alignment. There's a value add, that it adds to those processes, because the goals and the expectations are very, very similar. I think that's one. I think, we have a lot of people actually, who come to us and ask, have you considered using Temporal to implement some sort of, like blockchain yourself, or having a more tight integration there.

Right now, we don't have any plans for that, but I think it's something that you will likely see grow organically in the future, even if we don't have any hand in it. I think that's something that people are going to start realizing is a really good synergy and they want to start building those systems natively with Temporal or Cadence from the beginning.

Yeah. I think there's definitely a lot of overlap there. I think, they represent similar desires in different contexts of making sure that you can prove what happened and have some way to reason about it, and not feel there's some other element, or entity that can get in the way and mess with those things, so you don't have the visibility and clarity.

[00:47:49] JM: Awesome. Okay. Well, that sounds like a good place to close off. Thank you so much for coming on the show. It's been a real pleasure talking to you.

[00:47:55] RG: I've had a great time.

[END]