EPISODE 1472

[INTRODUCTION]

[00:00:00] SPEAKER: Conductor is an open source, microservices and workflow orchestration platform which originated at Netflix. Boney Sekh co-founded Orkes Inc, a company focused on offering Conductor as a service. Boney joins the show to discuss how engineers leverage Conductor to build highly reliable distributed applications using microservices architecture.

[INTERVIEW]

[00:00:21] JM: Boney, welcome to the show.

[00:00:23] BS: Thanks, Jeffrey. Thanks for having me.

[00:00:25] JM: You work on Orkes, that is the correct pronunciation, right?

[00:00:29] BS: Yes, Orkes. Orkes is our company.

[00:00:31] JM: Okay, Orkes, it's an orchestration system for microservices and workflows originated at Netflix, and I want to start the conversation with a discussion of some previous technologies that are in this area, microservices, and workflow orchestration. And when I think about micro service orchestration, the first thing that comes to mind is Temporal and the Cadence project, which is a workflow engine for long lived workflows. But I would classify that more as like a microservice orchestration system probably. And then there are systems for more complex workflows, like data driven workflows, that you have a family of Airflow, Prefect, and Daxter.

So, I think of this class of technologies of microservices and workflow orchestration systems as encompassing some of those different technologies, and those are the ones that people who are listening are most familiar with. Maybe we could start off with just your perspective on those technologies that are perhaps better known because they've been in the market for longer, and maybe you could just give your perspective on them, strengths and weaknesses, and we can use that to segue into a discussion of what you built at Netflix.

[00:01:51] BS: Yeah, sounds good. That's a great question. So yeah, workflows and the need for workflow engines have always existed, right? I think there's a ton of products out there that you called out. I was also listening to a podcast about union.ai the other day. That's also a very similar product in the space. But I think when it comes to a few of them, like you called out, they're very specific to use cases around the ecosystem of ML and AI, some of them. And then others are, like, there's something less common out there as well. I don't know if you mentioned that. But it's called commander, it’s also a workflow orchestration engine.

[00:02:26] JM: Okay.

[00:02:27] BS: So, some of those are like meant to run smaller scales of workflows, and especially based on some of the pricing models that we've seen on their portals. I think what makes Conductor different is a bunch of great features, it has. I think I can talk a little bit about that. But before that, maybe I should tell about a little bit of history of how Conductor came to be right. It has been around since 2016. So, I think it was around the time when Cadence first originated as well, in Uber. It has been around for a while, although it has been less popular, because I think we haven't done a lot of outreach or evangelization of this product as much as some of our peers have done. This is what we want to come in and do right now.

So, when it comes to features and makes Conductor different, I think the idea of Conductor is that it decouples the flow logic from the implementation, and I think this is unique about Conductor. This offers a lot of flexibility when it comes to configuring your flows and running it. Because your flow doesn't have to be coupled with an implementation, which means you can run different versions of your flows in production, you can have different variants of the same flow, you can run AB test by routing a small percentage of the traffic towards a new version of the flow that you have configured. So, there are all these features that makes conduct a unique and special.

Conductor also comes with like the visualization and observability tooling. So, you can visualize the execution before it starts. Also, each of your execution can be visualized after and you can see the path it took and the path it didn't take. A lot of the services that we talked about earlier only shows you the steps that actually ran, but that doesn't really fully paint the full picture, right? So, that what makes Conductor unique as well. There's a lot of tooling around the support as well. Conductor as a platform allows you to restart, retry, pause, resume workflows, you can skip steps. And these are all things that comes into play after your day one of development, like they do up until however long the service lives. And these are all like some of the features that makes Conductor unique, which I don't think I've seen in all the other products that are out there.

One unique power of Conductor is it's also designed to run besides your existing infrastructure, meaning it's designed to integrate with your brand feel applications, services that were already running in your production. Sometimes we call them as legacy services. Conductor makes it really easy to onboard onto those services as well, which means running your legacy applications alongside a modern infrastructure becomes easier. So, I don't know if I covered some of the points you were looking for. But I can go into any one of them in depth.

[00:05:08] JM: No, that's a great overview. I think maybe the next place we could go is to talk about some specific use cases. If there's a specific type of workflow or example workflow, a prototypical workflow, that would be good to cover, you've got a lot of different customers at financial institutions, infrastructure companies, Tesla, going after customers, but they're at least users of Conductor. So, maybe you could talk about some of the prototypical workflows that are a good fit for Orkes?

[00:05:37] BS: Yeah, Conductor, like you said, if you noticed, it's used in different industry verticals. I think we like to see it as a truly generic orchestration system. So, it can be used in the world of AI/ML, for example, or even for your business flows. But we try to focus more on the business flows as much as possible.

I can give you a few examples, maybe that's a good way to talk about the use cases. I'll start with a few things that we are using this at Netflix for. So, one of the big use cases of Conductor is how we ingest the media content that is coming from all of the productions that are happening for Netflix. Different studios across the world is producing content for Netflix and this is daily cuts, production files, all kinds of resources and outputs from every production process. These files need to be ingested inside Netflix. And this is built using a platform that is backed by Conductor. So, what happens is each of these files come in, and there is a ton of flows that need to happen. One file maybe just directly goes into an S3 bucket. But another file needs to go through an approval process or another file needs to trigger a big chain of processes that need to happen.

All of these flows are configured in Conductor and deployed. This is one use case that we see people using it for media workflow orchestration. Another use case at Netflix, this is actually linked to how the users use the Netflix UI. So, when you watch content on Netflix, there are issues that you run into. Subtitles may not be in sync with the audio, there are playback issues. These are things that you can actually report problems using the web UI of Netflix. What happens is we collect these reports, and there are really complex flows that needs to run, including human driven tasks to reach a resolution for these problems and uses reporting. Sometimes they run into like hundreds of steps. So, this is another use case where Conductor is used to configure those very dynamic nuanced flows for each type of issue that the users report and seeing that to a resolution.

All in all, there are hundreds of millions of workflows that runs at Netflix every month, and I can also mention a couple of use cases that we recently talked about in the meetups that happened in the last couple months. So, there’s tis FinTech company called Black Diamond from Advent. They use Conductor heavily across all of their workflows, and they talked specifically about this use case about how they use Conductor to track signatures for document using DocuSign. So, what happens is it spins off a dynamic workflow for the number of signatories required for a specific document, and it can keep track of if the signature is completed. Once it's completed, it can run all the configured post processing that needs to happen once the signatures is done.

Some of these workflows can run up to like months long, up to six months, specifically, because that's the limit that DocuSign has. So, this is an example of a long running workflow. And then one more use case I probably want to call out is like how Frames, it's a network automation company. They use Conductor as their platform and one of the things that they have done is they've created a UI on top of Conductor, and they allow their customers to configure the workflows dynamically. So, this is like Conductor powering that platform that Frames has built for Netflix for automation, but also extends that feature to the end users who can now configure different workflows on top of the functionality Frames offers.

So, these are some of the use cases we've seen. It's truly very generic and widespread, and then there are other scenarios where we've seen our customers use it for security, automation and things like that.

[00:09:11] JM: So, when you look at use cases, like the media side of things that you've addressed with Netflix, like a multi-step workflow of producing media files at scale, can you shed more light on what is required to build an orchestration system that fits those kinds of use cases, that contrasts with other kinds of orchestration systems?

[00:09:39] BS: Yeah. So, Conductor was created at a Netflix survey back in 2016, and it was primarily, you know, maybe this is a good time to tell you a little bit of a history of Conductor. This was when I spent time at Netflix, where I saw Netflix's second major pivot to be a company that produces content. The first major pivot was when they switched from DVD to streaming and the second one was when they decided they have to produce content themselves, right? Which means –the way that Netflix was going to do it was to do it globally, different languages, different countries. They all are very nuanced. They have different rules, different regulations, different directors, and producers. It does it differently. All of these was leading us to a place where we had to do a lot of engineering work.

So, there were a lot of microservices that were already available that had all the bits and pieces to get the job done, and it was pretty easy for us to build nuance for any specific API that was missing. But the real missing layer was how do we orchestrate these nuanced flows across all of these existing services? That's when Conductor came out. But before that, we were using an eventing architecture. So, it was Choreography. However, that wasn't very scalable. Because what we noticed was engineers started hand wiring these flows in dedicated services whose sole purpose was to manage this flow. What we were essentially doing was writing more services, that was purely meant for just orchestrating and putting – or, the other option that people were doing was to pollute existing services to do more things than what it was supposed to do. For example, they were thinking we were using the image encoding service to also deploy into the CDN and that was kind of polluting the purpose of that original service.

The thing at Netflix was the platform team had great tooling to launch new services, which means like new services were getting churned out often, because it was easy to do. But it came with a maintenance headache, and no one really knew how things were running. The choreography meant that the process was hidden behind events, and although we had tracing tools, it was very challenging to manage this whole thing easily. The production process that I mentioned earlier means every other day, we'll get a request that we need to change the flow of how content is produced in Korea by a little bit, because there's a new law, or a specific director wants to do this in a slightly different way.

So, all of these changes, we're getting really, really hard to manage and that's when we introduce Conductor. Because we try to invert this, from choreography to orchestration and all of these flows became definition or configurations on the platform. This meant there is a lot of visibility that you get into the flows that were deployed on day one, and this visibility also meant a lot of the changes that were being asked for, was easier to do, easier to change. New engineers who were tasked with making these changes were able to like see this visualized flow, which means, they were able to make those changes with less risk, and so forth. This became really popular inside Netflix after we launched and a lot of teams started adopting it. Netflix culture is unique in the sense that there's a lot of freedom and responsibility, so it's not like as an infrastructure team, we can come in and say, “Hey, here's a new way to do things, and you should adopt it.” But instead, it's something that we have to sell to the engineers and that's the sell process was really easy in this case, especially with the observability and visualization tooling that we were showcasing alongside Conductor.

So, one team adopts it, we show that as an example to the next team and they are quick to jump on it, and quickly, get adopted by many, many teams inside Netflix. And that's when we also decided that it could be beneficial to the community if you were to release this into the OSS.

[00:13:20] JM: So, when I talk to Temporal, for example, the Temporal workflow engine was incredibly hard to build, partly because like it's built for things like managing rides on an Uber like app or managing like a multistage banking application where there's financial dependencies. When I talked to Maxim, the guy who created it, a lot of what he said was that this was his fourth or fifth time writing a workflow engine. He had done it at AWS. He had done and all these other places. And it's incredibly work intensive. But I guess what I'm wondering is, is there a domain specificity to these different workflow engines? Do different companies, or different use cases want different workflow engines? Or do you think that these different workflow engines have largely overlapping functionality and utility?

[00:14:15] BS: I think when it comes to functionality, there's a lot of overlaps. At the end of the day, you're orchestrating your flows through these engines. But I think the way if I'm recalling correctly, Temporal uses workflow as a code mechanism. Basically, you write your workflows in code, while Conductor kind of does it differently, you use configurations for your flows, and it's also very simple solution, what Conductor is, and it wasn't that hard for us to build this out to be fair. This is essentially a task-based queuing system. It should use tasks. So, you configure a flow, it breaks it down into tasks, and whenever a task is ready to be executed, it should use it. It's essentially a scheduling engine.

So, it's a really simple solution for anyone who's looking to adopt this, to understand how this works, and over the years that it existed, we've kind of abstracted away a lot of problems that you run into when you're building distributed systems, such as handling transient failures, handling network issues, service unavailability issues, and then the whole paradigm of managing state when you're running a large decision tree as a workflow. So, all of this is taking care of infrastructure and it's a simple state management system on top of a scheduling engine that makes it happen and it comes with working your flows and things like that.

So, I believe one big advantage, or one thing that we've noticed people adopting Conductor says, it's really easy to understand how this works. I can imagine building this myself, but instead of doing that, when you use the source platform that does what I think is an easy way to orchestrate your flows. To answer your question, I believe it's similar in terms of outcomes for people are using it. It's just different the ways it operates and whatever makes sense for you, will be your choice. I think if you were to pick Conductor, you get some of those things that I mentioned, like being able to work in your flows very easily, running different. Because, in essence, what we are doing is we are taking this Lego blocks of code that you've built and deployed as microservices, and you're building flows on top of it. You're just orchestrating that flow using this configuration that you deploy on Conductor. That's the fundamental parameter of Conductor, I think, that makes it very attractive.

[00:16:28] JM: So, you mentioned that you listen to the episode about the data workflow orchestration system, union.ai. When you think about the design specifications for like a data focused workflow engine, versus a more general-purpose system like Conductor, are there specific parts of the system that you see as more appropriate or less appropriate in each case?

[00:16:54] BS: Yeah, so I've not used Flight, I think that was the intent that was back in the podcast that I heard. But I've used Airflow before. One of the things that Airflow comes with is this ecosystem of operators, that makes a lot of sense for your AI/ML workflows. So, you can kind of use these readymade functions that are there in your flow. So, that makes it unique or that makes it a better suitable engine in some situations. This is something that we are working to come to, in Conductor, we're trying to introduce all of these operators like functionality, which takes care of a lot of common things that people need, and that can be once we have enough of those, it may be a great fit for like an AI based, or an ML workflow engine. I don't know if that answers the question.

[00:17:41] JM: Yeah, it doesn't. Can you go a little bit deeper on that?

[00:17:44] BS: Yeah, for example, if you want to process like an S3 file, or things like that, there are ready made operators that are out there for some of these engines, which is like pre-made functions that you can import and start using. So, that gives a lot of advantage to these tooling when it comes to specific use cases, such as building workflows around ML or AI. Conductor on the other hand has always been focused on your business flows, your service flows, for handling your – for example, things like dealing with the signup requests from a user and doing all the provisioning for the user. These are like more API transactions that happens as a chain of API calls that you make or based on the decision tree. And these are use cases, I think, better fit for Conductor. On the other hand, if you have like a huge data processing workflow, which needs to talk to different databases, and so forth, sometimes using an engine airflow would make better sense.

[00:18:38] JM: Okay. So, if we're talking about multi microservice orchestration, the code that you're writing, like you're talking about provisioning resources, one service calls another to provisions and resources and then needs to be able to do something about it. It needs to do something with those resources. So, you have a chain of services that need to interact with one another. Is there like the notion of a promise, or some kind of callback system? I guess, I'd like to know a little bit about the way that you handle asynchronicity, and then you can also talk about just more generally distributed systems questions like, how do you handle service failure or service not responding to you, stuff like that?

[00:19:17] BS: Yeah, this is what Conductor does really well, right? You can configure these asynchronous flows that you have, which is very common in like almost any business that you operate, and you can have that orchestrated by Conductor in a very, very reliable way. So, the idea is that Conductor will look at your configuration, which is your flow definition, and it knows what's step one, what comes after step one, and where does it branch off into forks and where does it have to join back and all that. These are all defined as a configuration. And then the orchestration engine takes care of making sure all of those steps execute, and these steps can be very long running steps. You can have something that waits for months, as I mentioned about the use case of DocuSign workflow from Black Diamond. Or it could be very short running workflows, short running tasks that finishes in microseconds, and then continue to the next step.

So, the flow is managed by a Conductor, which means the decision of which step to run next is the one that Conductor makes a decision on, and it should use that particular task for you to execute. This task that you're executing, they are your microservices code. It can be a lambda function, it can be a service that you've deployed in a Kubernetes cluster, it can be a third-party service that you're integrating with. So, you can bring together all of these different types of dependencies into that single flow, and it has built-in features that allows you to retry it, if it fails and these are all configurable. You can decide how many times you want to retry and what should be the gap between the number of times you want to retry.

It also comes with features like rate limiting. So, this is a very common feature when you build systems, and when you're integrating with third-party services, especially, there is usually a rate management that you need to do. You can't like call 10 grids, email API a thousand times a second, because they may not allow you to do that. So, if you have situations like that, you can control that also through this flow definition and that takes care of all of that. So, Conductor basically has code and logic that kind of does all this functionality for you.

In terms of failures, it not only retries for you, but if workflow ended up failing halfway through, you can use the support functionality later on to retry or retrigger them, or ask it to resume if it is stuck in a – ask it to pause or if it is stuck in a failure loop, and you want to fix that problem before you want to resume that workflow. So, there is all of these features that comes with an orchestration engine like Conductor.

[00:21:45] JM: How are those retries and distribute systems resiliency systems implemented?

[00:21:50] BS: So, Conductor when you deploy it, it has a few things that it comes with one is a queueing engine. So, it uses Redis as his underlying queuing mechanism. There are different options that Conductor can be plugged into. The open source version of Conductor has different implementations for the queueing engine that people who use ActiveMQ and RabbitMQ for their queuing requirement. And then there is a state management store that you need along centers.

So, these are two things that it depends on. The state management can be again, there is a variety of storage options out there in the Oasis ecosystem, Postgres, Cassandra, MySQL, for example. So, combining the queueing engine and the state management engine, makes sure that your data is never lost, any tasks that you were supposed to run is always executed at least once, and that's how it manages resilience. There is like enough redundancies on the service layer, to make sure if one server of Conductor goes down, the next server that is living will pick up until the server that went down is replaced. So, all of that is kind of built into the architecture.

[00:22:54] JM: When you decided to turn this into a product, I assume that there was a lot of work to make this functional as a cloud product, you'd obviously worked on it at Netflix, and knowing what I know about how Netflix build services on top of AWS and does a lot of work around distributed systems, probably the system that you had inside of Netflix was very similar to what you've built as a cloud service. Is that accurate? Or has there been anything notably different as you've taken this to being a cloud product?

[00:23:27] BS: No, the foundation is Netflix Conductor in what we do have may be slightly different from the OSS is, some enterprises that we've been talking to has requested for like specific features that they want, like SSO integrations and role-based access control. So, some of these things are just the features that we've added on top that we eventually want to push to the Oasis as well. But that's the only main difference between what's running in the Oasis and what we have as a cloud product.

[00:23:57] JM: Are there anything when you think about building a cloud product more generally, we've had a lot of interviews with people who build various databases and other scheduling systems, all kinds of distributed systems problems on top of the cloud. Is there anything you can share about taking one of these distributed systems products to market in the cloud that you can share?

[00:24:18] BS: Yeah, that's a great question. So, Conductor always is right. I think I kind of told you earlier about how it has a different – it has various options for the dependencies that it has for the queuing engine and the state management. So obviously, not all of them are maintained to the point where you can run highly scaled workloads on it. One of the things we did was to pick the best choice. In our case, we picked up Redis as the queueing engine, and Postgres as our state management store, and we kind of tuned that up to make sure that you can run like scalable loads in production.

So, this is definitely something that we've done on top of what the Oasis offers. The other thing is being able to provision a cluster with a few clicks of a button, so that's what we've worked on since we launched last year as a company, and we've recently launched the product as well. What it does is it's very similar to like AWS console where you can go in and provision a database or an Elastic Search cluster, right? It's as simple as that. You go and click a few buttons, and you get the entire stack provisioned for you. A fun fact, is that we actually use Conductor as an underlying infrastructure for provisioning the systems as well.

I think, what you need to think about when you're building a cloud product on top of some of these things like Conductors, how do you address the different use cases that your customers have? So, maybe there is a customer who only wants to run a small amount of workload on this, and there's another customer who wants to run it like very large scale. In order to address that, we have different configuration options that leads you to slightly different setup, based on how you choose. And then there is a lot of auto scale features that we need to build for accommodating for spikes and how people want to use. So, these are some of the things we've taken care of when we built our cloud product.

[00:26:08] JM: I'd like to shift the conversation a little bit more to engineering around actually building Orkes. So, when you think about the different components of the system, if I'm building a workflow that touches multiple services. If I'm, for example, building a multistage workflow that spins up some infrastructure, and then one service loads, or pulls a video and transcode that video into five different places on a transcoder ladder. And then you have to add closed captioning to each of the transcoded sections of video, and then you have to, I don't know, run data analytics across all of them, or maybe even, that's pretty simple, because that's a pretty synchronous workflow. So, maybe you could give me an asynchronous workflow. But I'd love to know about what the developer has to write to build that workflow and what's actually going on under the hood to actually schedule that workflow?

[00:27:07] BS: Yeah, that's a great scenario that you used. I think we can talk about that one. So, you get a content, you kind of break it down into two blocks you encode it, and then you need to transcode and stuff. This is something that happens at Netflix. So, if you think about that, if you break it down, at the end of the day, say you go to the whiteboard and draw this out, it will be those steps, as you highlight it. You get the file. Now, you maybe want to break this down into smaller chunks, so you can process it independently and then each of the chunks get transcoded. And then you transcribe it, maybe for subtitles and things like that, and then merge it back together.

This process, if you think about how it works is you have all these pieces that needs to run to make all those steps happen, and then there is this gluing that you need to do to create that entire flow. It's a twostep process, right? One is you got to make sure that all of the steps you need are possible through some API's or through the systems that you have, which is like encoding, transcribing, and all of those aspects. Once you have those pieces, we can call them as these Lego blocks, then the next challenge is to come up and wire this up, and that's where you probably use an engine like Conductor.

With Conductor, you literally translate what you draw on the whiteboard as it is, and you say my step one is to get this file ingested. Step two is to create like a proxy for it. Or you want to try and break it down into chunks to process them independently. And step three is, for each of those chunks, you can do the encoding work, and so forth, right? So, whatever flow that you had painted in the beginning, is now what you would write as a configuration. There are multiple ways you can do that in Conductor. There's a way to write this in a JSON based data model, you're effectively representing your flow as a DAG, inside using JSON. And then the second way you can do that is to use like the SDK libraries that we have to compose that using code. And the third way, which is not out yet, is maybe to use the drag and drop UI that we are coming up with. You basically come up with that flow.

Once you have the flow, you persist that flow as a configuration in the Conductor platform and deploy it and start executing it. What will happen is the when the first file comes in, you want to trigger a workflow that runs, that particular flow that you've just defined, and you can see that getting executed by calling those Lego blocks. So, the way Conductor will execute that is by going through the sequence of steps you've defined and then calling the appropriate Lego steps that you've previously defined, and deploy it.

[00:29:41] JM: If you contrast that with a way of writing a multiservice workflow, without a workflow engine, what are you gaining by having a workflow engine?

[00:29:53] BS: Yeah. This was exactly what was happening when we first built Conductor, right? I briefly touched on it, like, what was happening is people were hand wiring these flows, meaning you kind of hand wire these exact steps that you would otherwise use Conductor for. The biggest advantage, or other disadvantage of having or doing that way is that your flow is hidden behind in that actual implementation, which means you can you can visualize it on how it actually gets executed. If let's say that was running in production for a while, a new engineer coming in would be very hesitant to change the flow that is kind of hidden behind in code. So, one of the things that you get out of the box with Conductor is these executions are which is visualized on exactly how it's defined and how it gets executed.

You can see that flow that you've previously defined for each of the execution instances that you trigger that workflow, right? Not only that, you can have different versions of it and this is where it gets really tricky if you were to do that using that hand-wired mechanism, you now have to have like a lot of default conditions, or if it is, for this condition, trigger this variant that we have written. If it's for that condition, trigger this other variant. But you don't need to worry about all that if you were to use Conductor, you kind of have all of that visualized and configured as a configuration. So, it is shifting that problem from the code complexity to like a configuration and its configuration is proven to be very effective in these situations when you have similar things, when you want the same code block to run, but you want to run it slightly differently. So, most systems tend to directly use configurations for that and that's what Conductor does for you.

[00:31:35] JM: You've got the core workflow engine built, you got a cloud service for it, where do you take the product next?

[00:31:44] BS: That's a great question. So, at this time, we are fully focused on doing two things. Two things, mainly. One is to evangelize and rally the community behind Conductor. And the second is to make it really easy for someone who's coming across Conductor from the moment they learn about it, to running a massive production workload. We are trying to optimize on that, trying to improve on that, trying to craft the journey that someone can take from when they first become aware of Conductor, all the way to running like critical production workloads in production. That's really where we're focused on right now.

Where do we want to take it next? I think we've thought about that a little bit. We think we can help with like specific verticals that are using Conductor, for example, for infrastructure provisioning, or for running like security related workflows and things like that. So, we want to pick an industry vertical, and maybe help with some of the common things that they often have to write themselves and maybe abstract away as part of the platform, is where we are thinking at this stage.

[00:32:48] JM: What's been your impression of the market? Are people formally ready to adopt workflow orchestration? I guess, workflow orchestrations as a concept has been around for a while. Maybe it's worth discussing, why does this field get continually reinvented?

[00:33:04] BS: Yeah. I think it's the ecosystem that's also evolving around it, like how you use systems, how you build systems. Traditionally, a lot of the engineering resulted in monolithic applications and it's only recently, you can see there is a shift towards microservices. Netflix was definitely a pioneer in this. It started doing this very, very early. But what I've seen in the market, it's only recently that companies are now adopting microservices, mainly, because a lot of the companies are moving from a data center served environment towards cloud. When you try and choose cloud, you kind of use a platform like Kubernetes, or something to manage your workloads, and when you use containers in Kubernetes, it naturally leads you to think about smaller services with all the support in the ecosystem that's available in Kubernetes, for example, service discovery, service mesh. With all of that features, it makes it very, very easy to break up your services into smaller chunks, and making it like microservices, and API's and API contracts is way more easier than like managing a large monolithic application, especially when you have a very large team.

So, I think the ecosystem around how you build applications has evolved over time, more teams are adopting microservices. And that's when you know you kind of run into this challenge of how do you want to orchestrate your flows around it. The legacy workflow engines were all very much centric on how you can run it on top of monolithic application like the BPMNs and all of those workflow engines that used to exist in the past. But with this microservice architecture, you got to choose between choreography or orchestration, and when you do that, you want to rely on a platform that can do that for you instead of handwriting it, and that's when there's need for an engine like Conductor comes out and I believe the timing is pretty much around the fact that a lot more people are now moving to cloud than before.

[00:35:07] JM: It makes a lot of sense. So, as you look at the different use cases, like I look at these use cases that you have listed on your website like food delivery workflows, patient management, patient medical records, workflows, telecom, subscriber and billing management workflows, there's really a lot of things. So, as customers adopt Conductor, or Orkes, are you seeing them refactoring existing code to include Orkes? Or is it more like just new workflows that they have? I guess, I could see them refactoring old code to use Orkes if the old code, for example, has like failover cases or something like that. But more generally, I would expect this would be like used for greenfield stuff.

[00:35:58] BS: Yeah, that’s an interesting question. I think one of the reasons why people look for a platform to abstract away things is like the problems that they run into, right? Especially those hand-wired flows, all of those, that's our biggest sort of challenge to or competitor, or if you were to ask me, it's the existing hand-wired flows that people manage.

The challenge with doing that is you have to replicate the functionality of managing resiliency, the reliability aspects, the state management across multiple different use cases, so you are kind of repeating that process over and over again. And that's when you kind of look for a solution that's out there, and most people that we're talking to, or we've talked to, are looking at migrating their existing services onto Conductor platform. And this is really one big advantage of Conductor, and I think, kind of touched on it earlier about how it was designed around the fact that we need to be able to integrate with legacy services or brownfield applications.

If I were to talk about that, if you just draw a circle around your existing service, and when you integrate Conductor, it's going to just live on the circumference at the edge. So, it doesn't really get deep into your code and you don't have to do any deep integrations. The SDKs or the client libraries that we publish are all very thin. It's just wrapping API calls for the most part. It makes it really easy to work with your existing legacy services. There are many situations where you want to integrate with that. Maybe there is a mainframe system that you use for processing transactions that you can't replace with anything else. For example, in banks, like JP Morgan, right? So, these are situations where you don't want to force people to do something new, but rather like provide a way for them to integrate their modern infrastructure to also work alongside their legacy system, and that's doable with Conductor’s architecture.

[00:37:51] JM: How active is the Conductor open source community? Is the project continuing to be engineered on?

[00:37:59] BS: Yeah. There is a pretty strong team at Netflix, that's their full-time purpose is to manage the Conductor, because it's pretty heavily used inside Netflix for a lot of critical workloads. They've been continuously engaging with the open source community throughout this time. Definitely answering questions that are coming through and any sort of support or feature requests coming from the community. Obviously, now that we also hear, we want to take that up a couple of levels. We definitely want to be a resource for anyone who comes across Conductor and wants to try and use it and help rally the community around it. So, if you've been doing a lot of meetups lately, that's been a new thing that was revived. When COVID came about, I think the Netflix stopped doing like in-person meetups and all that and we just restarted that. We still through like videoconferencing, but we want to soon start having some sort of in-person conferences around this.

So, we're really doing a lot more than before to rally the community. We have a Slack channel where you can ask questions, and there is a Discord channel as well. It's slowly gaining membership, and we hope to improve that over time. Just to add on, in terms of activity, there's quite a few pull requests and comments that are happening in the Conductor OSS, which has always been up to a decent standard, which we're also trying to make improvements on.

[00:39:27] JM: Are there any distributed systems use cases that are particularly challenging to orchestrate even with the help of a workflow engine?

[00:39:38] BS: So, almost all workflow engines by nature is designed to handle asynchronous flows. I think one of the things that people often could benefit from if let's say this engine went to do that is to handle like real time API calls. This is where you have a single API call that's coming in and it needs to essentially orchestrate a flow by talking to different services to gather data from different services, and to answer that call that the user has. This is something that it's not as straightforward to do with any of the asynchronous workflow engines today. And something that if any of the engines are capable of doing would be very, very helpful to the development community. So, this is something that we are also considering on how we can reuse the orchestration engine of connected to even power fail time called. And I think this is one scenario I can think of.

[00:40:30] JM: Does a workflow engine add or reduce latency to a net overall timeframe that takes to complete an operation?

[00:40:42] BS: Yeah, I think every workflow engine, like when you're offloading the state management and scheduling aspect to the engine, that comes with the fact that you need to be able to schedule first off and get that assigned to a specific task. So, it does come with that cost. But if you were to hand wire this flow, that cost would still exist, especially if you were to use some kind of queuing mechanism to make this happen for the sake of resiliency, right? When you compare with hand-wired flows that use queues behind the scenes to make the orchestration happen, it's almost identical. But if you were to compare that with someone making API calls, just in the code to make this happen, there's definitely that additional cost of having that engine orchestrate this flow for you, which is usually very, very negligible. It usually happens within milliseconds, the scheduling aspect and decision tree parsing. So, that comes at a very low cost, but it does have a price.

[00:41:42] JM: I guess, to wrap up, I'd like to hear your perspective on cloud infrastructure from a higher level. So, there's a lot of work around making distributed systems easier to operate, and Conductor is obviously one of them. There's AWS native stuff, for building your workflows. There's infrastructure as code. And I wonder if you have any perspective on, if the modern infrastructure innovations are laying the foundation for anything next, like what does the next wave of infrastructure management tooling going to look like given that we have additional leverage from simplifying these previously complex workflows?

[00:42:29] BS: Yeah, I think the ecosystem is definitely evolving towards a world where a lot of things are abstracted away for you. I believe, which leads you to write less code to make your functionalities work. And also, you kind of get things that otherwise you had to manage it yourself, things like resiliency, the reliability of the system, and all that is kind of abstracted away into platforms, and that frees the engineers up for dealing with a lot more business centric problems, what makes your business work and focus on that. I think that's a good place to be, right? I believe that's a place we are slowly going towards any sort of – I believe all the companies these days are really eager to offload functions that are not their core business to either platforms like us or other third-party sort of systems that they can use easily. I believe that's really where the world is going. You can already see that with a ton of things like sending emails or SMS or clicking payments. You tend to rely on third-party systems.

So, I think the world of building software systems in a distributed way is going to be based on – a lot of the things will be, you can start using out of the box by integrating with these third-party services, and I believe that's where it’s heading towards.

[00:43:56] JM: Yeah. I kind of wonder if there'll be more types of integrated platforms. I look at – I know one of your colleagues worked at Firebase for a while. Next generation clouds like Firebase and Netlify seem to combine a lot of these different tools and bundle them together and it’s simpler and easier to use, overall platform. I don't know, maybe there's something similar that will be done in data orchestration. I guess there's some work around that. But yeah, I agree with what you said.

[00:44:28] BS: Yeah, for sure. Definitely, when it comes to abstracting away that those common problems, I think that's where the theme is always going to be. When we talk to customers, we realize okay, there is this common function that multiple people want to run, so why not just abstract it away as a platform managed task, so you kind of get to write less code. I think that's where it is going, writing lesser code to manage complex flows, being able to change flows, without having to invest too much in engineering resources and so forth, is probably going to be the driving factors of how new systems are going to be built.

[00:45:06] JM: Cool. Well, Boney, thank you so much for coming to the show. It's been a real pleasure talking to you.

[00:45:10] BS: Thanks, Jeffrey. Thanks again for having me.

[END]