EPISODE 1043

[INTRODUCTION]

[00:00:00] JM: A workflow is an application that involves more than just a simple request response communication. For example, consider a session of a user taking a ride in an Uber. The user initiates the ride and the ride might last for an hour. At the end of the right, the user is charged for the ride and sent a transactional email. Throughout this entire ride, there are many different services and database tables being accessed across the Uber infrastructure. The transactions across this infrastructure need to be processed despite server failures, which might occur along the way.

Workflows are not just a part of Uber. Many different types of distributed operations at a company might be classified as a workflow. Banking operations, spinning up a large cluster of machines, performing a distributed CRON job. Maxim Fateev is the founder of Temporal.io and the co-creator of Cadence, a workflow orchestration engine. Maxim developed Cadence when he was at Uber, seeing the engineering challenges that come from trying to solve the workflow orchestration problem.

Before Uber, Maxim worked at AWS on the Simple Workflow Service, which was also a system for running workflows. Altogether, Maxim has developed workflow software for more than a decade. So he knows a thing or two about this. It was a great show with a highly experienced engineer in the domain of workflows.

[SPONSOR MESSAGE]

[00:01:36] JM: You probably do not enjoy searching for a job. Engineers don't like sacrificing their time to do phone screens, and we don't like doing whiteboard problems and working on tedious take home projects. Everyone knows the software hiring process is not perfect. But what's the alternative? Triplebyte is the alternative.

Triplebyte is a platform for finding a great software job faster. Triplebyte works with 400+ tech companies, including Dropbox, Adobe, Coursera and Cruise Automation. Triplebyte improves

the hiring process by saving you time and fast-tracking you to final interviews. At triplebyte.com/sedaily, you can start your process by taking a quiz, and after the quiz you get interviewed by Triplebyte if you pass that quiz. If you pass that interview, you make it straight to multiple onsite interviews. If you take a job, you get an additional $1,000 signing bonus from Triplebyte because you use the link triplebyte.com/sedaily.

That $1,000 is nice, but you might be making much more since those multiple onsite interviews would put you in a great position to potentially get multiple offers, and then you could figure out what your salary actually should be. Triplebyte does not look at candidate's backgrounds, like resumes and where they've worked and where they went to school. Triplebyte only cares about whether someone can code. So I'm a huge fan of that aspect of their model. This means that they work with lots of people from nontraditional and unusual backgrounds.

To get started, just go to triplebyte.com/sedaily and take a quiz to get started. There's very little risk and you might find yourself in a great position getting multiple onsite interviews from just one quiz and a Triplebyte interview. Go to triplebyte.com/sedaily to try it out.

Thank you to Triplebyte.

[INTERVIEW]

[00:03:53] JM: Maxim Fateev, welcome to the show.

[00:03:55] MF: It's great to be here. Thank you.

[00:03:57] JM: There's a type of procedure that occurs in software that is often referred to as a workflow. What is a workflow?

[00:04:06] MF: That is a very interesting question because different people have very different opinions on that. I personally think that workflow is practically any program which takes more than just request-reply practically, a program which has inherent state, which goes beyond a single request-reply, and you need to manage that state. It can be relatively short-lived, but it can be very long-lived execution.

Most people don't think about workflows this way. They see workflows as just sequence of step, and this apply to very small set of scenarios. But when you start thinking about workflows, it's practically any program which goes beyond single request-reply. You start seeing workflows everywhere. So you can think about like Uber, for example, of a trip as a workflow. You can start thinking about [inaudible 00:04:52] a driver as a separate workflow. You can start thinking as bike rental, like for example, if you look at JUMP Bikes as a workflow. As the same time, you can have a customer loyalty program when you – For example, if you're an airline and you have airlines points, managing those points is technically a workflow because you listen to events. There's state which [inaudible 00:05:14] for a longtime, practically for the lifetime of a customer. Then these states used to take some actions based on a number of points you took or based on time, for example, once a year, you need to give some present or nullify those points based on whatever business logic it is. Practically, almost every distributed business application from an infrastructure automation to – I don't know, events. Event processing can be thought as a workflow.

[00:05:38] JM: I love that definition, the idea that you can define it as something that goes beyond a simple request response. Architecturally, it sounds like a workflow is a set of services that are interacting with one another. This is just generally how we're going to be modeling a workflow. Would you say that is accurate? Is a workflow – Architecturally, is it a series of services that are going to be calling each other?

[00:06:09] MF: I think it's just one way to implement workflows because you absolutely can have workflow in a monolith. You can have single monolith which implements a lot of actions and these actions practically orchestrated by the workflow. They are just steps. Absolutely, these days, a lot of those actions live in the services. If you have service rent architecture, you absolutely need workflows or orchestrate operations there, at least operations would go beyond single request-reply. But you absolutely can have workflow in a single binary in this monolithic application because you need to manage your state over some period of time.

[00:06:45] JM: The example you gave of a JUMP Bike. I think this is a nice little example. I've rented a bike before. I walk up to the bike. I do something like scanning a QR code. The bike unlocks, and my session begins and I can go ride my bike around for a while. Then eventually I

lock up the bike somewhere else. The session ends and eventually I'm going to be charged for that. So that's a very long-lived workflow. Well, I mean, whatever. Depending on how long the bike ride is how you define long, but it's going to be 5 or 10 minutes at least of me riding this bike and eventually I'm getting paid. So maybe the workflow takes 15 minutes altogether and it's got some services that are calling each other on Uber's backend. Why does it even matter that I define this thing as a workflow? I mean, we know that there's just going to be some session of me riding this bike and it's going to be modeled on the backend somehow. Can't we just think of it as this collection of services? Why do we need this abstraction? This idea of the workflow to model it around?

[00:07:53] MF: I think there are multiple questions there. First is what is workflow from business sense? Not from how we implement that. I think independently how it's implemented, it is still a workflow, because you have multiple operations, which are tied together with some state, which go over to the time and a lot of those operations makes sense only in certain state of the process. You need to – Also, a big part of that is reliability. If you run these program over a long period of time, probably of computer is going down or having outages and infrastructure having issues, so new deployments going on. It's pretty high. You need reliability. You need to make sure that this the system is robust against failures. Independent of how it's implemented, you're implementing the longer running business process which need to be robust against the failures.

Then there are different ways to implement that, and what you described is kind of the way people do that. They just practically create [inaudible 00:08:51] of different components to provide that reliability. If you just don't use existing engine, like workflow engine, what you would do, you would just use queues. You would use databases. You would maybe use some timer service or just pull database for messages in certain state. You will kind of write to assemble that business workflow as a bunch of independent services. It's usually a pain to do that. Then if you have existing kind of infrastructure to do that, like workflow engine, then you certainly don't need to reinvent that wheel, you just implement your business logic.

[00:09:26] JM: And how did this workflow management cause issues at Uber? Because you originally built this or helped build this workflow engine called Cadence at Uber. What kinds of issues did it cause when you did not have a single system to help you manage the workflows?

[00:09:47] MF: I think the main problem it causes is that everyone reinvents the wheel, and this wheel is not very easy to reinvent. The way I kind of see it is that, imagine, there were times when databases weren't a thing. Every program would go and write and read to disk and write back to disk and keep records in certain format and re-implement all the necessary querying logic. Then databases were invented, and now if you're trying to build business application and you say, "No. I'm not going to use database. I will write – I don't know, records myself, and created them, and join them myself and maybe write the whole engine to do that." People will think you are crazy. I think we have the same right now with this type of applications, which are at least in business sense, workflows, that everyone reinvents the wheel. Everyone will try to come up with some system which supports the requirements, but then requirements will change or load in the system will grow or you need to add monitoring. You need to add capabilities to troubleshoot things in production. All of that, these are hard things, and we are reinventing the reinventions, reinventing them every time. This is exactly what Cadence tried to solve. It tried to solve these ad hoc workflows solutions, replace them, because now developers don't need to spend days and years kind of fixing the ad hoc solutions. They can just use existing platforms and practically just focus on their business logic and the platform would take care of the reliability and all other aspects of running high-load and highly reliable production system.

[00:11:16] JM: Okay. Great. Just to go a little bit more thoroughly into the high-level goals of Cadence, which is this workflow orchestration engine that you've worked on. What is the purpose of Cadence?

[00:11:30] MF: Purpose of Cadence is pretty ambitious. It actually redefine how you build distributed applications. Obviously, it's not all possible with distributed applications. There is a certain set of use case, which fit it pretty well. But there is very light set of use case which feeds it. Every time you have something like, as I said, which is workflow, which has state and state which lives beyond single request-reply, and it's not just [inaudible 00:11:55], right? Obviously, if you're saving information, written information, and showing it in the screen, there is not much of the workflow there. But if you're doing something meaningful, like for example, order processing or – I don't know, signup flow of customers, sign up flow support ticket and so on, it has kind of multistage state, and that is the problem it tries to solve.

[00:12:15] JM: What kinds of failures can occur? When you're talking about modeling these workflows and making them easier to work with, if we don't have a workflow orchestration engine, what are the kinds of problems that are going to occur?

[00:12:31] MF: Oh, let's look at the kind of simplest – I think let's start from the kind of basics. Right now, most of the code we write is kind of – At least pieces of code is planning. It's not related to your business logic. Think about like past when we had – I don't know, Microsoft Word Processes, which wasn't web-based. You would [inaudible 00:12:50] and then you forget to press save button and your computer crashes. You had to press save button very frequently. Right now, you type into the new version of Microsoft Word or Google Doc, you don't even have save button there, because it's automatically uploaded to the cloud.

The same thing with use cases, which require workflow because, practically, every time [inaudible 00:13:10] computer system, because memory of the computer system and the computer which executes it can go down anytime. Practically, what engineers do, they press the save button after every state transition in the code. Practically, you do something, you need to load state from database. You need to update that state. You need to save it back.

Practically, the first failure you have to deal with is actual failure of the process which runs your code. The way we do that is, again, a checkpoint in that state every, every time, and we do it manually. Practically, every time you get a request, you load state from database. You update that state. You save it back. Then we can reply back to the customer. For example, it's even worse. If the system used to make those stream call to another system, if this is not a simple call which like can time out in a millisecond, if it can take potentially long time or if its system is down and you need to keep retrying for a longtime, you probably need to checkpoint your state that practically you intent, or I'm going to call that system. Then you need to [inaudible 00:14:05] records in the database saying that this is time out of my call. If I crash, load that and verify this call company to denote. Then you need to make a call. Then you need to practically asynchronously for the completion and then completion comes. You will need to process that result. Again, loading state from database and processing that and saving it back. Also, you need to pull on these timers, records, to time out the call. This is just one call. But if you need to [inaudible 00:14:31] retries, if you do multiple calls, if you need to chain those calls, this logic becomes extremely hard.

Practically, the main type of failure which engineers work with is a failure of the actual process, which runs the code. Then obviously as we are living in perfect world, then you need to deal with failures or other systems. For example, if you make a call, that call can fail. Those stream systems can be down. It can be down for a very long time. It can be down for hours or even days. That is another type failure vision you need to deal with. Then you need to run – In some systems if you get, if you get error from some API call, you need to rollback, because we don't have a distributed transactions anymore. People use what's so called sagas. Practically, you make calls to multiple services, and if one of them cannot complete your request, you need to go compensation calls to other services just to roll back the changes which were already done. You need to make sure this is complete. So these type of failures also need to be managed.

[SPONSOR MESSAGE]

[00:15:34] JM: When you need to focus on building software, you don't want to get bogged down by your database. MongoDB is an intuitive, flexible document database that lets you get to building. MongoDB's document model is a natural way to represent data so that you can focus on what matters. MongoDB Atlas is the best way to use MongoDB from the company that creates MongoDB. It's a global cloud database service that gives you all the developer productivity of MongoDB plus the added simplicity of a fully-managed database service. You can get started for free with MongoDB Atlas by going to mongodb.com/atlas.

Thank you to Mongo, DB, and you can get started for free by going to mongodb.com/atlas.

[INTERVIEW CONTINUED]

[00:16:28] JM: To play the devil's advocate as to whether we need an entire workflow orchestration engine to manage this, why not just use Redis, for example? Why not just have this resilient in-memory system that can store all of my state about the transactions, the long-lived transactions, the workflow management that's going on across my system? Can't I just use Redis or etcd for that matter? Why not just have a lock server that manages the information that I need to know about this stuff? Why do I need an entire engine to help me through these kinds of workflows?

[00:17:12] MF: Practically, I don't think it changes anything. If you have database or user IDs, right? It's just kind of type of database, right? Just fast. It's still the same. You still need to load that information when you execute your business logic from that external source. In this case, would be Redis, and then you need to – Or etcd or whatever. It's still database in whatever format it is, persistent storage. You need to load this state from that storage and save it back on every state transition. Managing that state is non-trivial. Again, I didn't even start talking about timers, timeouts, retries, all these things, which are not trivial to implement. Then how to scale those things, because it's pretty straightforward to partition your data across business [inaudible 00:17:52]. If you have millions of your customers, you can go and partition that across multiple database instances using [inaudible 00:17:58], but it doesn't work for times, for example, because every time you make a request, you need to create a durable timer.

Durable timers are not as easy to partition about because they kind of partition in different dimensions. If you start thinking about all aspects of reliability and scalability, it's not that trivial to build systems which support those requirements. Again, people do that. It's kind of the status quo right now. But my point is that they actually spent disproportionate amount of time building kind of not their business logic. Like look at for example some business logic. Let's say you implemented in a single process without thinking about this process going away. That logic can be like hundred lines of code. Okay, call that API, call that API. Change arguments to another variant. Call that AP. Then based on whatever load, you so something else. But then I say, "Okay, now run this for million for customers and make it robust against all possible failures." Then this code will become monstrosity of callbacks, which is very hard to understand and manage and has very little resemblance to the original code.

[00:19:02] JM: There are some other tools that are popular that could be described as workflow orchestration. Airflow comes to mind. How does Cadence compare to Airflow?

[00:19:14] MF: First thing is that Airflow tries to solve kind of the same problem, but it has very, very different approach. Also, it tries to solve that problem sets for very strict subset of use cases mostly because it was created around data pipelines. It has very limited requirements on terms of scalability because you cannot just go and say, "Okay, I'm going to use Airflow, for example for a bank, to transfer money from one account to another." Because if you need to do

a few thousand transactions per second, I don't think Airflow is technology which you can rely on. That is kind of first part, is that it never was built for very high-scale. It was built for data pipelines.

Second part is I think the biggest difference is in the way workflows as I described. The idea of cadence is very different from all existing workflow engines in a sense that – Let me kind of just start from the beginning. Imagine that you have computer, which has fully fault tolerant memory. In the future, we probably will have RAM which will obviously be durable. So it will completely change the way we write applications. Because if you eliminate exactly what I said, eliminate this cycle when you need to load state, update the state and save it back to database, because we can just receive request, update your local variables of your program and return request and your local variables will be preserved because memory is fully fault tolerant.

The same way, if you do downstream call and this call takes 5 days or requires retries and everything, you still can be blocked in the same line of code for a very long time because, again, that memory of your computer including stack, all variables, everything is preserved. That will be a very nice world to live in. It will completely simplify the way you write applications. This is not magic. This is exactly what Cadence does. It allows you to write normal code, just code as you would write in your preferred programming language. You just write code, which updates local variables [inaudible 00:21:02], makes API calls and everything, and then this code will – It's abstraction, but it gives you – When you write code, you write code as your memory cannot be lost, including local variables, stacks and so on.

If you talk about Airflow, and practically all other existing workflow engines, they now execute your code directly. What they do – Even if they are codebase. For example, Airflow is codebase. Not like DSL-based or JSON-based as most other engines. Airflow – All code does instantiates DAG. DAG is some kind of – You can think of this, a very limited version of abstract syntax tree of your program. Then what [inaudible 00:21:35] use the actual DAG, which is very, very limited version of kind of programming constructs.

What Cadence does, Cadence actually executes code, like code in your program. This code doesn't instantiate some other representation of program. It is actual code. What Cadence gives you is this virtual memory abstractions. [inaudible 00:21:54] changes the way you think,

because you need to rethink about new abstractions. Your abstractions is still code. If you need to do if statement, you will just write if, else, normal programing language statement. If you need to loop, you will just do normal loop. If you need to do whatever business logic of your application, you will do business logic application. If you are a Go programmer, you will get channel and you will listen on the channel and select on that if necessary to process external requests. You do need to kind of rethink the way you model your programs.

The best part is that [inaudible 00:22:22] about complexity, because there is a reason if you don't use DSLs, like syntax, abstract syntax trees or JSONs to write millions lines of code. If I come to somebody and say, "Okay, take this complex business application and rewrite it in JSON," I don't know, "Linux Kernel." That would be insane, right? Why? Because you cannot write a very complex business logic in this high-level languages. You need like a real programming language because they have enough abstractions and enough capabilities and enough learnings to deal this immense complexity. Practically if I say, "Here's the Java program or Go program. Write me – I don't know, hundred thousand lines of code," I'm pretty sure that you can write them in a way that programmers can make sense out of them. Because Cadence, again, is just code, complexity of the completion you can write using this platform is practically unlimited and it's very, very different from any other workflow engine.

[00:23:13] JM: Great. What are the steps for defining a workflow in Cadence?

[00:23:19] MF: For defining workflow, practically you just – It certainly depends on the programming language you use. Currently, we support out-of-the-box Java and Go, and Python client actually, which is external contribution is almost ready as well and no [inaudible 00:23:32] client is also an external contribution. Let's talk about Java, for example, just to start. In Java, what you would do, you would create an interface. You will put method there, which you have to notate as workflow method on that interface and it's just normal Java interface otherwise, and then you implement that interface and you just write code inside of that interface.

The biggest limitation in how these – Why workflows are different from any other code is that workflows cannot call external APIs directly. They always have called them for so called activities. You can think of them as tasks as well. We call them activities. Practically, if you need to send message or call some external API or whatever, you always put that code into activity.

Practically, all the external calls go to activities. Then workflow makes API calls to activities for special API, which practically activities that also implement interface. Practically, I'm kind of jumping ahead of me. If you do activities, you just implement Java interface and every method of that interface becomes an activity.

Then inside of workflow, you get special stops to that interface, practically, which implements that interface and you make calls to those and it can be [inaudible 00:24:40] calls. Practically, you say my activity implements – I don't know, deposit method and withdraw method. Inside a workflow, your call activity don't deport, like stop of that activity don't deposit, put whatever parameters are there. This call can block for five days. Then it unblocks, you go to next line, which can't withdraw, or whatever. Practically, you just write code which calls into the activities. Then you have other – All features of the language available. Also, you can receive external events, because workflows can react to external events, which in case of Java would be just callback method, which is called every time the external event is sent to the workflow.

[00:25:18] JM: Great. I want to continue to move through the different vocabulary. You have activity tasks, decisions, decision tasks, activity workers. Just to make sure people who are listening are following along. What cadence is doing for the developer is it's providing this environment that is durable. It's a durable environment that can help you manage this workflow that may have state that needs to be acknowledged for days or months at a time. That's a long-lived session. If you want to maintain a session for that long as a workflow session, you're going to need some kind of durable, flexible environment that gives you the paradigms you need to be able to call these different services over this long period of time.

[00:26:15] MF: There are two parts there. One is the programming model. How do you write code? Then a separate path is how you deploy that code and how you run that. When you write code, you don't need to think about that. The only requirement is that you use appropriate workflows APIs to do certain thing like get in time and also you use activities to call external services instead of making direct calls. What a framework does is it preserves the state of your workflow code including call local variables, state of your [inaudible 00:26:45] and everything.

Then you need to deploy that and run that. The way Cadence does it right now is that you have backend service, which exposes API, and right now we are switching to GRPC API. Practically,

it's a GRPC service, which behind-the-scenes consists for multiple components, but these days it's just a bunch of Docker images, practically one Docker image which you can deploy in multiple roles. It also requires a database.

We currently support MySQL and Cassandra. Postgres is also a contributor, but it's not production-ready yet, but it will be production-ready soon. In general, almost any database which supports multi-row single shard transactions can be used potentially with Cadence. Then the biggest difference of Cadence from other workflow engines, that it doesn't execute actual code of the customer, even workflow code. For example, in Airflow, you kind of send that deck for engine to execute. In case of Cadence, what workflow code and activity code lives outside of that service. This code is part of the customer service. It's the same way as you can think as Kafka consumer doesn't belong to Kafka cluster or publisher. It's just external process which communicates to the cluster. The way Cadence code, like workflow code an activity code lives inside of worker process which connect to the service externally, and this worker processes are done – Link with APIs. Practically, client-side libraries, SDK. For example, if you're doing Java, you will just include Java SDK as a dependency and then you just can program against this API. Then you need to run those workers. These workers are external.

These workers, what they do, they connect to the service, and service internally has queues. They just pull for tasks from those queues to receives tasks. For example [inaudible 00:28:29] activities not in work directly. It puts a task in the queue and then activity worker picks up the task and executes that and then reports back to the service about its completion of failure.

Nice thing about, that workers are out of the workflow control. So it means that workers cannot be able to reload it by rate of requests. They cannot process requests fast enough. The request will just be back-logged in the appropriate queue for those tasks. Does it answer your question or probably you need more detail there? Because I can go certainly for a very longtime describing that in details.

[00:28:59] JM: No. It's great. I want to examine this from a number of different angles. In order to give a different angle, I'd like us to talk about the Cadence architecture. We can talk about the architecture of Cadence itself, the Cadence service and the backing storage system. Then we

can talk about how an individual workflow might integrate with this service. I think we've given the listeners a decent overview for what purpose this thing is solving.

Let's talk about the overview of the Cadence architecture itself. Can you tell me what is going on in the Cadence service and what the runtime looks like?

[00:29:36] MF: Okay. I'll try. It's certainly 5-hour lecture. Let's start from the service API. Practically, the core service API, to workflow, is to send signals to them, which are external events, which you want to deliver. Practically, when you initiative a new workflow, for example, you press a button on your website, for example, to initiate money transfer. You will call start workflow execution operation on the service API. That will create state for that workflow, and you also pass business ID, because Cadence is a fully-consistent service, it guarantees uniqueness of workflows by ID. It usually will be – I don't know, transaction ID or customer ID. It depends on your use case. In this case, you will create that object inside of Cadence.

Also, what it does, I didn't mention that, but the way we reconstruct state, we event sourcing. We record every event related to that workflow. With every workflow maintains so-called event history. Practically, when you start workflow, we will workflow executions static event into the event history for that specific workflow instance. At this point, this call returns. Because service accepted that, this is it.

Then what happens is that as workflow call doesn't live inside of this service, we need to call into the actual code. How it's done? There is a worker which contains the workflow code, which lives outside the service, and this workflow worker lives in a so-called task list, which is essentially queue with some name. This name practically can be chose in their application.

What happens when you start workflow, it's not only creates a state and not only appends event in the history. It also creates a task in that queue, in that task list, and all of that atomic. It is very important, because Cadence eliminates a lot of race conditions. Like when you implement ad hoc system, for example, you use queue and you use database. You always have a race condition. What do you do first? Do you update queue? Do you put message in a queue and then update the database or you update database first and then put message in a queue?

Whatever order you choose, you are wrong, because unless you have transaction in those two, you will end up with bunch of race conditions there.

Cadence has absolutely guarantee when you call start workflow, it will update the state, create the history and also create a task. If it fails, none of that will happen. Then this task is put in appreciate queue and workflow worker picks it up and executes workflow logic, which for example say executive activity, and it doesn't call that activity directly. It just sends command back to the service saying, "Okay, I want to execute activity A." What it will do? It will update the state with the workflow inside of the service. It will append event to the history saying – Practically, comment to the history saying activity task scheduled. Then it will atomically also create activity task in appropriate queue, which activity listens on. Activity listen to their own queue.

At this point, what we call decision task from the workflow worker will complete. Then activity worker will pick up the task from that queue, execute it and then call complete activity back to the service. Service will add event to the execution history, which is activity task completed and call workflow again delivering the new event to the workflow and it will go back and forth like that.

The complexity on the backend is – Practically, what it needs to do, it needs to maintain practically unlimited number of those workflows because you want to scale out as wide you need, billions of them potentially. Then for every state transition of the workflow, you need to have a timer, durable timers. Every time you call, for example, activity, you start multiple time, you start a timer. How long can it run? Activity can hear beat, so it might end up creating the heart beat timer as well. You can have multiple activities running in parallel. You need timer for each of them.

The whole workflow has its own timer. Also, a business logic can ask to sleep for some time. So you need timers created from the business logic. That is another part. Also, because problem is that you cannot do it naively in terms of scaling, because if you shard by workflow, and this is what we do internally. We do shard by workflow ID. So it can actually have a lot of database instances, or like Cassandra nodes, and then we can scale out. But then it doesn't help these

things like delivering these queues, because you can have single queue delivering messages to activity – Fleet of activity workers.

This queue is sharded completely differently. Then you need to have durable timers. So sharding durable timers is a little bit orthogonal to both of those. Most of the complexity of the service is around how you show that you can scale all of those different dimensions and then make sure that this system is robust and also fully consistent. Yeah, and all transitions are atomic.

Okay, it was long, I think, but –

[00:34:01] JM: No.

[00:34:01] MF: It's certainly a very incomplete description.

[00:34:04] JM: Totally fine. I mean, we're not going to be able to cover the entirety of Cadence in this podcast. We just want to give the listeners a glimpse into what this does so they can learn about whether it might solve a problem they have. You've mentioned that the data in the Cadence servi9ce is backed by Cassandra. Why Cassandra?

[00:34:23] MF: I think there are multiple reasons of that. One is that Uber just had hosted Cassandra service internally. It's kind of has like internal version of more like RDS of Cassandra. That was probably most obvious choice, because our team didn't have to maintain that. Also, in general, Cassandra is pretty nice in a sense that you can add capacity relatively easily. It deals with resharding and all of that. You can just add nodes and it runs.

Unfortunately, because we are a fully consistent service, Cassandra has a lot of limitations around consistencies. We had to use lightweight transactions, and they certainly not the most optimized and loved feature of Cassandra. But it took us very longtime to get a try. Don't try it at home, please.

[00:35:05] JM: Okay.

[00:35:05] MF: We later added support for other database. MySQL is one production already, and we will have more of them. I think this time we can support almost every database out there which supports transactions.

[00:35:17] JM: Got it. A workflow is going to involve, likely to involve multiple services, or I guess, as you said at the beginning of the show, it could just be a single monolith. But for each service or monolith that is going to be interacting with this central cadence service, tell me about what the communication is. You mentioned that these services would embed a client library. But I want to know more about what the client library does and what the communication protocol is between these individual services that are being orchestrated together and the central cadence service.

[00:35:55] MF: There are actually two very different models there. One model is that your service uses Cadence individually and communicates with external services for activities. For example, if you have existing RPC services, not kind of workflow-enabled, which usually just have very short requests, like short-time amounts for their requests and you need to call them. You still want to use workflow, because these services can be down and can have outages, or you need to run compensations like sagas. You call these services. It can be pretty fast, but then one of them is down. Maybe you want to retry. I've heard one requirement from one of our customers that some don't stream dependency, which is third-party, said, "We cannot subscribe to SLA. We can be down for three days." They say like, "[inaudible 00:36:44] we can be down, but three days." Then people come back to us and say, "What do we do? This is like three days. We cannot retry from Kafka," they were using Kafka back then, "for three days, especially if it's subset of messages. How do we do that?" Because Cadence supports exponential retries of activities out-of-the-box, we practically said, "Okay, set your retry [inaudible 00:37:03] for three days and you're fine."

But in this model, what happens is that your service host the worker, which host both activities in workflows and this workers connects to Cadence for GRPC, but it's kind of just happens inside of the library so you don't need to manage that in any way besides giving correct connection stream. Then this service will execute activities and workflows and these activities will deal with failures of external services and retries and so on.

Another model, which is I think is more interesting, is that you can think about Cadence as a service mesh. A service mesh for longer running operations, because right now if you think about it, all existing – We're all talking about service meshes, but it's always around short-lived operations. It's practically about request-reply. Yes, there are also – Once we deal with queues, they practically allow you to configure queues and receive message from queues. But technically, if you think about it, most of this is about request-reply.

Inside of Cadence, workflows can call other activities, and workflows like synchronous. When you look at the call, it's synchronous request-reply. You make request and then you wait for it in block-in mode and then you get your reply. The different is that this request can take 5 days or one year. It doesn't matter, or maybe you just have retry policy for three days. So this request will block for three days until request goes through.

What you do, if you model other services as activities or child workflows as workflows inside of Cadence, so each service will be separate deployment. Both of them will connect to the Cadence server. But then when they call each other, they will call each other as activities or child workflows and Cadence will become kind of this longer-running operation service mesh. That is the idea, is that you can treat it as a service mesh for longer-running operations and you can create service [inaudible 00:38:48] architecture around that. That is I think the kind of different way to think about it.

[SPONSOR MESSAGE]

[00:39:01] JM: As a company grows, the software infrastructure becomes a large complex distributed system. Without standardized applications or security policies, it can become difficult to oversee all the vulnerabilities that might exist across all of your physical machines, virtual machines, containers and cloud services. ExtraHop is a cloud-native security company that detects threats across your hybrid infrastructure. ExtraHop has vulnerability detection running up and down your networking stock from L2 to L7 and it helps you spot, investigate and respond to anomalous behavior using more than 100 machine learning models.

At extrahop.com/cloud, you can learn about how ExtraHop delivers cloud-native network detection and response. ExtraHop will help you find misconfigurations and blind spots in your

infrastructure and stay in compliance. Understand your identity and access management payloads to look for credential harvesting and brute force attacks and automate the security settings of your cloud provider integrations. Visit extrahop.com/cloud to find out how ExtraHop can help you secure your enterprise.

Thank you to ExtraHop for being a sponsor of Software Engineering Daily, if you want to check out ExtraHop and support the show, go to extrahop.com/cloud.

[INTERVIEW CONTINUED]

[00:40:35] JM: As you're telling me about Cadence and just how many pieces there are to making sure this works properly, I wonder what the testing process has been for developing Cadence. What kinds of tests did you put in place and what was your testing process to make sure that this workflow engine that is going to be applied in so many different ways works as you expect?

[00:41:04] MF: It took time. It just takes times. Just to give some background, both leaders of the project, me and Samar, we worked at AWS and we worked here. Samar also worked in Azure, and we build more than one project together. When we build in Cadence, we're building it as an AWS service from the beginning. We build it as like a hardcore infrastructure component. We spent a lot of time and a lot of REST testing and we have a lot of release pipelines, which run through practically as possible scenarios or using this service and we also run long-running tests. Our release pipelines requires I think at least two, three days to run to ensure that we don't have regressions.

Obviously, there is no magic. Sometimes, bugs slip through that. For that, we have staging pipelines. [inaudible 00:41:52] of what we had, we had customers running their staging workflows on staging, and we are doing upgrades of the staging environment as a test, first, for make sure that our upgrades don't break existing running workflows and also to ensure that systems stays healthy.

Also, even if in the case that bugs slip through production, we have a lot of protections and a lot of ways to deal [inaudible 00:42:15]. We practically never lost workflows. They could get stuck,

but we would find it out and solve that problem. But in general, our system was very reliable in production. But yes, it took a lot of effort and a lot of discipline to do it right.

There is other thing about testing, which I wanted to mention, is that because you write code, you can actually easily test it. You can test your workflow using your preferred unit testing framework. For example, in Java, you can just and use JUnit and you can write workflow and you work every activity and we're using Marketo or whatever preferred framework you use and you can write practically full-blown normal Java Unit test of all your business logic. The best part is that the unit testing framework that Cadence provides supports times keeping.

What I mean by that is, for example, your workflow does some iteration and then says, "Sleep for three days, because I need to do something three days later. Maybe send customer message." That unit testing framework will detect that the workflow is blocked for three days. It will automatically roll time forward for three days instantaneously. Then it means that your workflow unit test will run in milliseconds even without you changing any timeouts or timers, which says days or weeks, which is a very powerful feature for unit testing or for longer running business logic.

[00:43:30] JM: I'd like to run through another example, maybe in a little more detail about how an activity or a workflow would actually be implemented in terms of the different vocabularies, this activity, activity task, decision, decision task. Can you give an example of a workflow and put it in terms of programming paradigms of Cadence?

[00:43:54] MF: Yes. Practically, the terms you mentioned, most of them actually don't need when implementing workflows. You probably need them when you want to understand how the system works. On a high-level, you have activities, you have workflows, and workflows can be started and you have signals, which is events you can send to the workflow. These are practically the main abstraction you need to deal with.

All others are more like implementation details. It's nice to understand them and nice to understand how system works for troubleshooting, but when you write code, you just write workflows and activity tasks. The main rule is that activity task, external APIs should be called through activities. Practically, if you need to do money transfer, you will write activity, which will

actual bank API, bank APIs, and then workflow will just call those activities through activity interface stuff.

Practically, inside of Java code, you will say new interface stuff for this interface class. It will give you instance of the stuff which implements that interface and you will just call method on that activity directly and it will be normal API call. The difference is also that you need to understand that arguments as realizable, because you can plug in serializer. By default, we use JSON. Basic idea is that this should be [inaudible 00:45:03] types, because this is practically remote call when you call an activity. But these are main abstractions, that activities and workflows. Workflow code is just code, which again just has some restrictions how it runs.

For example, if you create multiple threads inside of workflow code, you have to use APIs provided by the workflow, and there are some other restrictions around, like for example time. You need to take time using workflow APIs. Basically, just calls, which are calls into activities and also you can call activity asynchronously, for example, and it will get promise back and then you will just can block on that promise. So it can wait full promises so you can do like asynchronous completion of that promise. There are a lot of kind of standard Java things, and this promise is practically similar to Java completable promise in terms of functionality.

[00:45:48] JM: Tell me more about the process of building this at Uber.

[00:45:52] MD: One thing is that just to give, again, some background. I was tech lead for the simple workflow service at Amazon, and some [inaudible 00:46:00] there as well. I spend a lot of time thinking about this problem. I'm doing this for probably more than 10 years straight solving this type of problems.

[00:46:10] JM: Wow!

[00:46:10] MD: At Amazon, I cannot like disclose a lot of internal information. It took us long time to get the first version out. It was multiple iterations. I wrote at least three client-side libraries and the one which Amazon has right now for this simple workflow service. Not many people know about it. There is actually AQS simple workflow service out there, which kind of implements similar ideas that Cadence does. But obviously, it was 10 years ago. It took us

some time to get it right, and it's still I think we didn't get it right. Especially the developer experience wasn't good enough for this service to pick up, I think.

Later, when we had this similar problem to solve at Uber, we kind of decided to not copy a simple workflow, but use the ideas that simple workflow had, but iterate on them and make it better. We build relatively simple version of that internally and they iterated – But a completely different backend and different APIs, because for example, at Uber we use Thrift over our own APIs. Then we are wrote client-side library in Go. Then we iterated for some time just to make it production-worth. We put in production. Then we started to kind of getting feedback from our users and the multiple, multiple iterations of the product. We started to get a lot of very important features.

For example, we support multi-region replication. You can have multiple clusters, which for application level protocol, talk to each other. Even if you lose completely the whole data center, you can continue you work, workflows container from a different data center out-of-the-box. That was very hard to implement feature and took long time to get it right. But most of the important services at Uber would never use this service, which doesn't sustain the data center outage.

The one thing which was great about Uber, Uber allowed us to build this project as open source from the beginning. Uber was very open to – You see a lot of full open source projects coming out of Uber because Uber had very, very open policy towards open source. If you had projects which could be interesting for the community, build it. Not even release. A lot of enterprises, you build something then they allow you to release. It's almost impossible to release it after, because usually dependencies creep-in. But because we built Cadence from the beginning as open source project, I think we managed to keep those dependencies at baby. Also, it helped us to get obviously external community.

[00:48:17] JM: When did you get the idea that a business could be built around Cadence?

[00:48:20] MF: I think it wasn't the motivation to build business around that. I think one thing we ship, I believe, is that the new programming model Cadence promotes and supports is very powerful one. I believe that it can change the way a lot of applications are written in the industry.

When we started Cadence Project, we always wanted to – That's why we wanted to make it open source. We wanted to make it successful not only inside of Uber, but outside.

At some point, we started to realize that being inside of an organization even as great as Uber is was limiting, because we couldn't really focus on external customers, because we just have different priorities, right? You cannot walk at Uber and do some features, for example, supporting Postgres database, which Uber doesn't use.

At some point, our project started to become popular outside of Uber, and were approached, told us kind of that there is possibility to get money and also they kind of helped us to understand that we really cannot make this project successful, really successful outside of Uber unless we can focus on that 100% of the time on external users. I always believe that the technologies there, and this kind of convinced me that my co-founder, Samar, is that we want to create a company around that and focus all our attention on making this project great.

[00:49:35] JM: The fact that there is not a de facto best workflow engine that is as general as you are pursuing with Cadence, do you have any ideas on why that is and why it's possible today, or is it just the fact that this so hard to build and nobody has put in a decade at workflow engines like you have?

[00:50:05] MF: I think there are various reasons for that. First, yes, you have to be stubborn. [inaudible 00:50:13] walked on me on a simple workflow, when I told him that I'm building a company around similar ideas said that I'm the stubborn person in the world he knows. Just people underestimate the complexity of these engines, especially if you need to build high scale one. That if you look around, there are quite a few. But if you say, "Okay, how many of them can support thousands of workflows per second? How many of them can scale to hundreds of millions of parallel executions or potentially more?"

There are not that many. I don't if probably the conductor would be the closest one from Netflix. I understand that the no-code is older age now, but I think the idea of no-code or low-code is about helping professional programmers to write to kind of automate what they're doing, and I think this is great, and I think it's super useful. There are a lot of actually local engines already

running on top of Cadence, but I think that is misplaced when we try to move programmers into that world. Software engineers love code and they just don't like to write plumbing.

What are we are doing, we're eliminating the plumbing component and allow them to focus on their business logic using what they like and this is their call. I think the combination of these just code first approach, and now the way we can call it, we call it workflows as code. This is kind of our view how to code. What we are doing is like workflows as code. That is one thing. Second is just having people who build AWS level services before and can spend time, thanks to Uber, for example, to make them production-worthy. This has been in production for three years at Uber. So that combination factor is not that easy to come up with.

[00:51:48] JM: And if somebody is listening to this and they're thinking maybe this can solve a problem that I have and they're thinking what is the best way for me to actually use Cadence today to actually try out this workflow system that you've built. What's the best way for them to start using it and to start getting their workflows orchestrated more properly?

[00:52:13] MF: Right now I would recommend to go to our new company website called Temporal Technologies, so temporal.io. We have links to documentation to GitHub repos and samples and everything from there. You can find all information about it there. Also, we have a link to the Slack channel and I'm always there, so you always can find me there and ask. Don't go implement complex application without getting design consultation.

One thing about these workflows as code is that they are code, but there are certain design patterns [inaudible 00:52:46] this time. Unfortunately, I didn't have time to write [inaudible 00:52:49] at least even blog about them yet, but it's nice to talk to somebody who wrote or at least witnessed a bunch of those designs, validate that you're doing the right thing.

[00:52:58] JM: All right. Well, just one more question, I like to focus this show around particular technologies. But you've been in the software industry for a pretty long time. You've been at Amazon, Google, Uber, Microsoft. You're at AWS. Now you're at your own company, and we are in the midst of this unprecedented change to the world. Obviously is unsure about what is going to happen with the changes due to this virus. But do you have any advice for people who are

navigating their careers in the technology industry right now that relates to this dramatic change that has occurred?

[00:53:36] MF: I think we are extremely lucky being in technology industry right now, because I think they are the least affected. I understand that still, especially if economy goes down, it will affect everyone. But obviously being able to walk from home without getting the same paycheck is a blessing. I certainly feel very bad for like the rest which cannot enjoy the same situation.

If you're in IT industry, I don't know. My feeling is that just become good at what you're doing. We are hiring right now and I'm pretty sure there are a lot of companies which are hiring. We certainly have the best job security right now. In terms of learning, I don't know. Just pick your area. Become proficient at that. From other point of view, people who are actually generalists and they are successful. But I'm certainly not one of them. As I said, I'm very stubborn. I've walked on the same infrastructure level problems for all my life.

[00:54:25] JM: Maxim, thanks for coming on the show. It's been a pleasure talking to you.

[00:54:28] MF: Yeah, thanks a lot for inviting me. It was a pleasure.

[END OF INTERVIEW]

[00:54:38] JM: Apache Cassandra is an open source distributed database that was first created to meet the scalability and availability needs of Facebook, Amazon and Google. In previous episodes of Software Engineering Daily we have covered Cassandra's architecture and its benefits, and we're happy to have DataStax, the largest contributed to the Cassandra project since day one as a sponsor of Software Engineering Daily.

DataStax provides DataStax Enterprise, a powerful distribution of Cassandra created by the team that has contributed the most to Cassandra. DataStax Enterprise enables teams to develop faster, scale further, achieve operational simplicity, ensure enterprise security and run mixed workloads that work with the latest graph, search and analytics technology all running across hybrid and multi-cloud infrastructure.

More than 400 companies including Cisco, Capital One, and eBay run DataStax to modernize their database infrastructure, improve scalability and security, and deliver on projects such as customer analytics, IoT and e-commerce. To learn more about Apache Cassandra and DataStax Enterprise, go to datastax.com/sedaily. That's DataStax with an X, D-A-T-A-S-T-A-X, @datastax.com/sedaily.

Thank you to DataStax for being a sponsor of Software Engineering Daily. It’s a great honor to have DataStax as a sponsor, and you can go to datastax.com/sedaily to learn more.

[END]