Build streaming integrations with relational databases using the JDBC Alpakka Connector

IBM Code Tech Talk

February 07, 2018

https://developer.ibm.com/code/techtalks/build-streaming-integrations-relational-databases-using-jdbc-alpakka-connector/

>> MARC-ARTHUR PIERRE LOUIS: Thank you, Kathy. Good afternoon. My name is Marc-Arthur Pierre Louis, I moderate these tech talks. It's a pleasure to have another one of these. Today is interesting, because we want to talk about streaming integration with relational databases using the JDBC Alpakka connector.

The Alpakka connector is streaming connector, and nonblocking which allows you to have more reactive websites and more responsive websites.

To talk about this solution, we have our friend from Germany, Markus Eisele, who is the director of advocacy at Lightbend. He is going to talk about index in the presentation because he is all over index. It's a pleasure for me to welcome you to this session about streaming integration was relational database and to talk

about it, here is Markus Eisele.

>> MARKUS EISELE: Thanks, Marc-Arthur and welcome everybody, wherever you are, like I'm sitting in my basement, which is pretty much located close to Munich in Germany. It's always fun to actually give those kind of on-line presentation.

I don't know if we should talk about index first, or last. But I intend to deliver my talk first. So you guys came here to actually listen to my talk, I need to share my screen -- and make sure you can see my slides.

Streaming integrations with Alpakka, before I actually start talking about Alpakka in today's technical topic, I want to make sure I have a couple of minutes to introduce myself.

My name is Markus Eisele, which is German name, I've been around the Java ecosystem since many years by now. I work as the director of developer application at Lightbend. I've been working around and with Java technologies for uncountable years. It all started when I actually was just a normal software developer at a German SI.

I had to work my way all the rungs up until I ended up being a principal architect, enterprise systems that are run by automotive and insurances in Germany and my hunger to actually share my knowledge and spread the word about lessons that I learned, that never stops.

I started speaking at conferences and actually [inaudible] most

recently in Germany, called Java land.

(audio breaking up).

If you want to be at a Java conference and also want to make sure you can have some great rides, yeah, you are listening, it's correct. It's happening in a theme park. If you want to ride the black bomb ba like the big developers do, that is the conference for you.

I have also been named Java champion a couple years back, I guess and I've been around Java ecosystem for quite some time. You can find me on Twitter at my fear and that is a story for a long beer. Whenever you hit me up at a conference ask me about that nickname and we will be all set for the next hour.

We want to make sure to talk about Cloud and microservices and streaming and why everybody is actually talking about that. It's the next big thing, everybody wants to re-architect their application towards microservice, we are talking about Kafka and infrastructure like message brokers.

The question is why are people doing this. Monolithic applications do a good job of the over the last couple years I haven't had a big problem. I build some the biggest monolithic applications you can think of based on Java E for example. What is the problem with software development these days?

This is a screen shot from result of a survey that we run recently. Biggest problem in software development these days, high infrastructure cost. People don't want to pay that much money for

unpremised solutions in their basement. I get that. Awful downtimes, like Canada getting overwhelmed on black Friday, that is only 9 percent. Meeting demands, that is a interesting one, I'll get back to that one.

Release frequency also 20 percent of the people, yeah, I get that. Monolithic applications, those things tends to be very dense, so project teams of 200 people, I think that was one of my biggest ones, but even on average, you have 20, 30 people working on the solid Java E enterprise application and you have month long test cycles. You have more than 500,000 lines of code. You have another 8 million lines of code in PL sequel hidden somewhere. You can't release that monster every other day for not even speaking about hours, right, so it takes a couple of months to get stuff like that into production.

Honestly, I have to admit just looking at the EU and European Union these days, there is a centralized legislation. The EU actually defines the new banking metrics that banks need to report on, the stuff needs to be in production less than six weeks.

I actually worked on systems that weren't capable of being deployed more than once or twice a year. Honestly, believe me my absolute most favorite deployment times have been December 24, or the day exactly before my vacation started.

Release frequency, that is a big motivator for me personally to switch over to microservices. But it's also that developer

velocity kind of part that is obviously like 40 percent, that is the biggest chunk here. What does that mean? That means people don't work together like looking at teams, but also involves methodologies. We are all looking at Agile methodologies, all looking at quicker round trips, but that also involves code.

The whole technical parts of the application, code is unreadable, unmaintainable, it's a big spaghetti kind of monolithic applications.

It also talks about systems and the complexity inherently built into those monolithic applications.

Let's dig deeper into meeting demands. This is a nice chart I actually found on the Internet. It gives a very good projection about the Internet users, like connected people to the Internet.

This grew significantly over time. There is no indication that it's going to stop, especially not when we are looking at how many different devices we are using these days.

Every one of those devices is actually going to be part of an Internet connected scenario or application use case, right. So the phones are the easiest part, fitness trackers, but also cars, just think about a reasonable 2018ford model and I'm picking that brand for no other reason than I like them and drive them.

They emit 25 gigabytes of data every day, just driving these cars. They use that data internally to steer the systems. But imagine what happens if those cars actually start to expose parts of those

data via the connection back to either OEM or insurances.

So we are also starting to see a lot more use cases of heavy data usage, so that goes to the point where we are no longer able to just process incoming data in batch chunks, and update on them over weekend. We need to start processing this data in almost near realtime.

The fun part is, getting millions of cars to send teeny little detail like the GPS position and the actual speed to an insurance, for them to compare that with existing maps and speed limits, for example, so they can start offering you new policies, depending on your driving behavior, like if my mom drives, she will pay at least 500 bucks less a year than if I drive cars. Those business models evolve from existing ones. Those literally, the only limit to those business models is your imagination and data protection laws in Germany. But that is another story.

Another interesting angle is that you start answering questions on data that is already recorded or captured, that you never even thought about, while building and designing these applications.

So you need to find better ways of crawling through lots of data chunks in your applications, and making those accessible to modern applications. All these are motivations that led to a shift of everything that we have seen in the past couple of years.

Mainframes are still part of a lot of critical things, but they for sure don't scale towards mobile devices. Decentralized

appserver infrastructures with various data centers connected via area networks did a decent job over the last ten years maybe. But they don't really scale that well anymore.

I think the largest Java E cluster that I ever saw in my past in history had 500 nodes. That is just not enough, if you just think about use cases like Twitter, for example. They wanted 500 nodes of Java E infrastructure in their basement, right, so they need to be way more flexible and way more scalable.

And all those developments over the last couple of years led to that distributed Cloud scenario. That is another part of the story, which is OnPremise Cloud. Why should I scale my applications to the worst scenario that I could even think of? Speaking of black Friday sales, right, so why do I have to put 100 servers in my basement just for that one week or month or whatever it is, when I only need one for the rest of the year.

Having that Cloud pay for what you really use model as the leading model for your resource consumption is also a big driver in that whole microservices and distributed systems thinking world.

Ultimately, I have to admit all those developments led to more users, more data, and way more systems that need interconnect and talk to each other, right? So at the end of the day, this is what we have to deal with as software architects, as software developers as of today.

That means we have to start to scale the most common programming

task ever. We have to find a way to make taking input transforming it into something, and producing a reasonable output, fit into the world that is insanely data intensive and highly asynchronous, right?

Streaming stuff for distributed systems is exactly what we want to do. I hear somebody in the back calling for functional programming. Yeah, that is definitely part of it. That will help, but it won't solve all our problems.

So just by looking at data as infinite streams, we won't solve our problems. We also need to take some other considerations into aspect here, especially flow control. We need to make sure that downstream slower subscribers aren't overloaded by faster publishers. There needs to be some kind of back pressure involved, so that the individual components actually are able to signal if they are overloaded or not.

We want to make sure this whole package of flow control works, instead of just dropping messages, if somebody has like an overflowed in-box, or worst case, maybe even an out of memory exception at some point, if one process can't handle what we are sending in terms of messages.

This is where streaming comes in. With the streams for Java developers is a really, just a initiative to provide standards among the libraries to encompass those efforts around JVMs to make sure they understand streaming, back pressure and flows. You can read

everything about it at reactive-streams.org. There also a link at the very bottom of the slides that you can definitely refer to.

I consider those little links homework. There is not a lot of time in those presentations to tell you everything about all the topics, and since my girls joined school they are both 6 and 9 right now they also start hating homework like I do. But I want to make sure you have a little bit of homework to do when you go through the slides and the recording afterwards.

The link here is a easy introduction to reactive streams for Java developers. Don't miss out on that one.

The idea of reactive streams is to have some kind of notion of shapes for reusable integration. Instead of trying to pipe an input to output stream which is not possible, we want to make sure that we have some shapes that can do some transformation on flowing data, without actually having to cast or do any unsecure things in between, or even without having to handle back pressure, and flow control in between components.

This is a very simple example how you can actually convert an integer into a string, and print it out on the system out. But how does JDBC come into all of that?

So yeah, when people talk about streaming and reactive applications, they easily mix up the fact that streams are a first class citizen in reactive applications, and one of us are not built towards that kind of model. On the other hand, what we are really

aiming for is modernization. So tearing apart the most important parts of those monolithic applications that need more scaling than classical centralized architectures can provide, and enable them through microservices, through reactive microservices, and I'm not saying that everything has to be a microservice-based architecture, but at least the most critical parts need to be strangled out of those existing monolithic applications.

Honestly, full disclosure, even if you have two stacks next to each other that are still kind of communicating and reactive microservice stack is taking over all the scaling, all the heavy loads from for example mobile applications, and the classical application is still used for your back end and customer service support, that is totally legend but you still need to find a way for those application parts to work together.

JDBC has a big advantage, at least in the Java world, it is broadly used. It is well-known. It is the reliable way to access the base in general, and it will always be involved in your modernization efforts.

Even if we are all striving for the ultimate and perfect design of our microservice architectures, and we also of course strive for decomposing our databases, but my assumption is that there will always be some kind of at least read only communication with existing databases, maybe even more. Maybe we even have to write in some kind of transfer table, some information that are coming in.

Another motivation for picking JDBC in that talk is that we are not only looking at microservices and modernization of applications but we are also looking at scenarios where we are moving away from these ETL style batch style applications structures.

Just a example, like I'm not going to name a specific insurance, but some insurance in Germany I work for, they had a specific window on the specific weekend late in December where they actually had to create all those invoices for their car insurances. And those batch windows with more customers with new business models gets smaller and smaller. So a weekend won't be enough anymore.

Some kind of magic needs to happen to existing batch and ETL architectures to make them more streaming architectures, to shorten those batch windows to literally zero, and give customers the opportunity to actually solve their problems on the fly, working on the thought data stream. That is another big motivation for picking JDBC, because instead of choosing any fancy no SQL or log based persistence mechanism, we can still use relational databases, even in streaming architectures, right?

Looking at JDBC, it's just not a good fit in a way, especially not for reactive applications. It's inherently blocking, the number of required threads is just way too high, very slow and long running operations tend to block system threats and you can't do fancy things like simultaneously access more than one database at once. It's just not possible. That is not the way most of the JDBC

drivers are built.

I have to admit there are a custom, there are a couple of specific JDBC drivers that support synchronous processing and are not blocking. But it's not the majority and it's always been a specific logic.

This is where finally, the Alpakka comes in, right? The Alpakka is obviously an, animal. But it also is a reactive enterprise integration project. You can find Alpakka, it's literally a number of connectors that allows you to not only stream in the reactive stream sense, internally to syncs but also interconnect various systems via different connection formats. Looking at the list we can read from a file and push something up to Amazon, or to the IBM Cloud. We could push data into mongo DB and reading from JDBC data source. This is what Alpakka ultimately gives you as a feature set.

To put that into a little more context here, we are finally trying to walk through a little bit of an example, so what you are going to see is the Alpakka connector which consists both out of Slick which is a open source framework for reactive JDBC, and the Alpakka Slick connector itself. We are going to use reactive streams implementation and http to expose endpoints, akka, http.

I think, one more slide, couple of dependencies. I actually want to walk you through the IDE. Let me see if I can pull up my IDE and share the right screen with you. Marc-Arthur, can you see my IDE? Nobody is listening? I'm still on?

>> We can see.

>> MARKUS EISELE: Awesome, thanks, Clyde.

Let's start at the very top and beginning. What we want to look at is the XML [inaudible] classical cluster Java setup for enterprise developers. Yes, you heard about license, you heard about Scala and you will stumble about stuff like Scala version in artifact I.D.s. Don't be afraid. Really don't be afraid.

This is something that we need for binary compatibility. But you don't have to deal with that in your day-to-day work. It just is part of the artifact ID.

If you scroll down, you will see a static name and property here pointing to the latest Scala version. We have an Alpakka version. I'm including the akka stream Alpakka Slick connector. I'm including theakka streams base library and http end point. I'm using H2 in the example, because I don't like databases that much. I wanted to have a very simple and reproducible test set to work with.

So I'm using a local database that actually puts all the needed file system information into my target folder of the Maven project and I do have a couple supporting scripts here that drop the tables of the database when I start the Maven compile and run process. They also actually create the table that I'm going to use in this example.

Drop, that is what we already looked at. I wanted to look at the tables. Yeah. As you can see, there is no magic in here. I

want to remind you that there is a link to my GitHub project which contains all the source codes. You can definitely fork it, give it a try, play around with that and follow up with questions.

What am I going to show you tonight is something very funny. Let me get my --

>> MARC-ARTHUR PIERRE LOUIS: Markus, sorry, my chat obscured my microphone. I got it out of the way. The H2 driver, H2 that you were showing was database or what was it?

>> MARKUS EISELE: Say it again, please?

>> MARC-ARTHUR PIERRE LOUIS: H2 that you showed, was it a database?

>> MARKUS EISELE: Yeah, yeah. That is just the free database, H2. Give me a second. Let me go back, that is a free open source database.

>> MARC-ARTHUR PIERRE LOUIS: Cool.

>> MARKUS EISELE: It's pretty straightforward and simple (overlapping speakers).

>> MARC-ARTHUR PIERRE LOUIS: I went into the no SQL, mongo and IBM no SQL database [inaudible] good time to go back to try.

>> MARKUS EISELE: Honestly it's a good choice in many cases, especially for reactive applications. But again we are looking at the other side of the story here. Right? So looking at like legacy integration, stuff that is already there, and yeah, back in the days when I started programming, nobody talked about no SQL at all.

Let me, do we have some questions while I'm playing around with my stuff?

>> MARC-ARTHUR PIERRE LOUIS: No, I see one thing that popped up, let me see if it's a question.

>> MARKUS EISELE: Wonderful.

>> MARC-ARTHUR PIERRE LOUIS: It's not a question. Okay. So keep going.

>> MARKUS EISELE: I just actually want to start that little project here.

(ping).

That should work. Does it or doesn't it? Let's see if that works.

>> MARC-ARTHUR PIERRE LOUIS: It's compiling. (overlapping speakers).

>> MARKUS EISELE: Still trying and it takes a little. But we will get there. This is a live demo, that isn't recorded, so it has to have a little bit of like [inaudible] feel to it. All the dependencies should be downloaded. No worries, so what I'm doing right now is obviously execute my Java environment, and I did put the code haus Maven plug in to call the main method on the DB processor class, which is the main class that we are working with. (overlapping speakers).

Yeah.

>> MARC-ARTHUR PIERRE LOUIS: There is a question for you. It

says can you please tell us about the licensing and usage of this project?

>> MARKUS EISELE: Alpakka, I'm not 100 percent sure but I believe that it's Apache 2 licensed so it's open source. It's community driven. So you just need to go to GitHub.com, Alpakka. Let's look. GitHub.com Alpakka, there we go. And you can, oh, my Internet connection badly needs an upgrade. But yeah, we will get there. Local host. Let's see what that fancy application does on the local host at least.

Wonderful. We are waiting for everything. The application fired up in the meantime. Our server is on line on local host. We can at least see the sample application doing something. What the application is doing basically is something that I probably should have shown you in the Power Point. I'm not sure if I dropped that slide. I did. Wonderful.

So, start-up server programmatically inserting 50 user objects into that user table, and this is actually a text field, and HTML index file which populates and connects to Web sockets back end, to stream those data out of the database, and there was another end point in this project which is called more and if I call that end point, I'll add another 50 users.

If I refresh the main website here, I'll see 100 users being inserted. Yes, I didn't pay a lot of attention, the name obviously wasn't, or not even the ID was unique. So you see a couple of

duplicates in here. You will see the reason for that very soonish.

Alpakka, let me look for a license, contributing, maintainers, license. There we go. It will get there. Licensed Apache 2. So whoever wants to contribute, this is definitely community licensed. It's, yeah, contributions and additions are just a pull request away, it's a open available repository, so call to arms.

Let's look at the user object first. This is not really dramatic in any way. As you would have expected, it just has an ID and a name, because I just wanted to have some kind of users in here.

So let's walk through the main DB processor, which is doing all the magic in this case. Can you see if I actually enlarge my screen? Is that also streamed? Or is that just me locally?

>> MARC-ARTHUR PIERRE LOUIS: Your screen is launched, yes, your IDE takes up the whole screen.

>> MARKUS EISELE: Wonderful. The first thing we are going to do is something that I haven't talked a lot about. Alpakka obviously has something to do with acka and acka streams. Akka is an actor implementation on the JVM. You can build amazing things with it. But it also is the implementation base for both akka streams and Alpakka. It's kind of a chain.

Everything we are talking about here is implemented on the actor model. You don't have to deal with the concept of actors, but everything you do with akka and especially with Alpakka starts with an actor system.

Those kind of two lines are mandatory here. The materializer is actually taking the flow and produces an output out of this flow. What else do you see? Couple of loggings stuff that I definitely always need in my applications, and the next thing that you find here is a so-called slick session.

This is very slick specific, but very well documented. There is an application conf, and what you have been missing in the Maven in the code so far are definitely the DB connection properties, right. This is something that Slick also needs. But in this case, we are actually reading it from the conf duration object which is passed out of this application conf. So that is this one line of code.

>> MARC-ARTHUR PIERRE LOUIS: Quick question, do they do connection pulling? I remember when I was doing JDBC a while back, connection pulling was a big thing. Do they do that?

>> MARKUS EISELE: Slick is doing way more. Slick is actually doing asynchronous JDBC, and this is why we are using it for JDBC through Alpakka. It does all kinds of asynchronous things that JDBC usually don't do. As I said, Slick is also a open source project, and I definitely recommend to check it out. You will find a lot of Scala examples out there. But yeah. Color me surprised. You can use it in Java based applications too and it's not a big problem.

>> MARC-ARTHUR PIERRE LOUIS: All right. Before we go to the next line of code, there is a question I'd like to take. How actively

this project is being developed, maintained? Also how do we get support for project in case we use it?

>> MARKUS EISELE: Which one are we talking about? Alpakka? Alpakka is community effort. (overlapping speakers) Alpakka is a community effort. And there is no supported offering available right now.

Support is pretty much community-based, so there are a couple of GitHub channels and mailing lists and everything you can think of. I think there are a couple of licensed employees in that project, at least engaged from time to time. But yeah, the projects maintainer is not a [inaudible] at the point, so no commercial offering and no support.

Requests are welcome, so if anybody wants to contribute, I'm happy to see that, I'm the director of development, don't forget that.

>> MARC-ARTHUR PIERRE LOUIS: Of course.

>> MARKUS EISELE: No questions?

>> MARC-ARTHUR PIERRE LOUIS: Not yet.

>> MARKUS EISELE: The next thing that I need is some kind of data. This is the random list of users that I'm generating here. Makes use of lambdas, probably seen that in Java code before, we are just creating a list of random users. There is a reason why actually the user I.D.s are duplicates, because I don't check.

Another thing that I wanted to extract a little more and make

the code readable is I created the function for the insert statement that Slick source that we are actually going to use.

(ping).

This is actually going to use, so to make it a little easier, it takes the user object and it puts it into the insert sequel code, right? We have touched briefly in the context of reactive streams, and [inaudible] same kind of construct. All you have to include are actually those sinks and sources. Sinks are places, objects where you push something. Speaking of database inserts, you are inserting users into the database. So you need a sink.

If you want to get something out of the database, you actually need a source, right? So you want a user stream coming out of slick statement.

This is exactly what you see here. The select statement, created, and assigned to a source.

Let's go down to the main method. This is where streaming gets a little bit tougher. First of all, we are actually going to do some cleaning, housecleaning. We need to make sure that the Slick session is closed at the end of our actor system. Whenever our actor system terminates, we want to make sure that Slick database session is also terminated.

This is literally what we register here. What we also do, we invert the first 49 of 50 users into the database by calling the inserts users graph runnable graph. This is a special construct

in akka streams. Instead of defining a sink and source we can also define an intermediate step like a runnable graph, that can be executed multiple times.

After that, we are actually going to start our akka http server and for convenience reasons, I put that same class into the DB processor. It defines the endpoints, the routing, so we have our data end point which is the Web sockets end point. We have our rest based more call which leads to inserting another 50 users into the database. We have the initial slash request that gets the index that HTML filed from our resource folder.

That is really pretty much everything we have to do here. In that case of funneling the data streaming the data into the text field, Web sockets based, everything we actually have to do is take our user stream and map this user stream object, the string value actually, to a Web sockets text message.

While I hover over here, you see that there is every object part of the Java DSL from akka http so not a lot new to learn, stuff you should have already seen somewhere.

We are just returning a handle Web socket message method here, but returns all the data streams from the sink that we actually created.

There is a couple of screen shots here that I don't really need. Give me a second. I need to switch back to my presentation.

>> MARC-ARTHUR PIERRE LOUIS: As you are switching, there is a

question. It says I see the idle time-out in the config file. Is that needed to be that high, five minutes? I tried the sample application.

(distorted audio).

Get time-out error. The question is the high five minutes in the config file.

>> MARKUS EISELE: That is actually the akka http server idle time out. If you have it up for more than five minutes, nothing happens. It just shuts down. But I have to lie about that. I would need to Google. I'm at my fear on Twitter and if you reach out to me, I'll be more than happy to answer that question off-line.

>> MARC-ARTHUR PIERRE LOUIS: Thank you. You can reach out to Markus at myfear on Twitter.

>> MARKUS EISELE: Or send me a E-mail. I'm m at lightbend.com.

We have done the code walkthrough. One thing that we have been missing and stuff that I didn't really want to dig into here is the so-called enterprise integration patterns. You might have heard about Apache camel, and if you have heard about it and you have seen what I have showed you right now, there might have been some deja vu maybe. As a matter of fact, yes, it's true, like Alpakka and especially in combination with akka streams is your streams based nonblocking integration toolbox.

So you can pretty much implement all the integration patterns, all the enterprise integration patterns with akka streams and

Alpakka. This is something that gives a whole different meaning to both projects, because like all of us had to work with these before. I found it very interesting to have a streaming approach to those integration patterns.

And I think I'm finishing with a famous, may the source be with you quote, in the greatest sense of "Star Wars."

I do want to leave you with a couple of next steps so you can find the project side of akka on lightbend.com. I pointed you to the GitHub repository, don't get me wrong, akka is open source but you can have commercial support for it. The only exception is Alpakka so the integration connector, this is a sheer community effort right now, a link to the documentation. Feel free to send PRs, raise issues, ask questions, more than happy to follow up whenever something comes up.

Our CTO and cofounder wrote a nice book about reactive microservice architectures. This is a free O'Reilly book. If you want to download and learn more about reactive microservice architectures, this is something you can take with you on your next long flight or with you on your next weekend, and because Jonas is CTO and I'm not and he wrote the book and I didn't, I had to write another one, and if he is talking about architecture, I had a talk about developing reactive microservices.

This is your follow-up. And there is another framework I didn't point you to which is called logum, something to help you implement

reactive microservices on the JVM. If you want to learn how to do that, that is a good starting point.

Last but not least there is something out of my history. Modern Java uses design patterns. If you are looking into breaking out monolithic applications and want to have more ammunition to talk to your management about how and when and where, this is another free wonderful O'Reilly book.

Thank you so much for your attention tonight, I hope it was comparably entertaining and I can show you something new that made some, sparked some interest in getting stuff up and running and playing around with akka streams and Alpakka.

If you have questions, I'm more than happy to take more. Otherwise, you can reach me at myfear on Twitter or send me a E-mail and M. @ lightbend.com.

>> MARC-ARTHUR PIERRE LOUIS: Some people commented how they enjoyed the session. Good job. Thank you for that.

>> MARKUS EISELE: Thank you so much. I think we are still up to talk about index, right?

>> MARC-ARTHUR PIERRE LOUIS: Yes, exactly. Before we get into index, I have a question for you. How, can you tell me the usage of this Alpakka connector into streaming analytics?

>> MARKUS EISELE: Not right now. In theory, we are not talking IBM product, are we?

>> MARC-ARTHUR PIERRE LOUIS: No. Not at all.

>> MARKUS EISELE: You can use akka streams and the Alpakka connectors for literally all low latency streaming problems. If you find the right connectors for both sites, like source and sink you can fire those as main method, have it as the stream processing engine fire hose in the middle, and the stream right away.

>> MARC-ARTHUR PIERRE LOUIS: Okay. In theory, you can do it. But you don't know about implementation right now.

>> MARKUS EISELE: It depends on the endpoints, right? I think I showed the available end points in one of the slides. If you find the right end point, and even if you don't find them, implement them and commit them. Send a PR. We are more than happy to take that in account.

>> MARC-ARTHUR PIERRE LOUIS: Excellent. All right, people on the call, if you want to experiment and come up with some great design and implementations, Markus promised that he will take a look at them. Right?

>> MARKUS EISELE: I will definitely do. I'm not sure I can approve them. But the akka team will definitely be there to judge them, and we are very very very happy to have more community contributors. (overlapping speakers).

Let's talk about index. My last couple of weeks have been comparably index-y, what is that, purplish? So I've been appointed the program chair which is a big honor honestly. I've been working in many program committees before. But index for sure was the most

challenging so far.

It has a broad variety of topics, and it's sheer and purely technology centered.

It's really, really interesting, as a conference from it for people to attend, it's 3 days. It's almost 170 sessions, including keynotes and workshops. It doesn't really cost a lot, right?

So can you point your browser to index conf.com? The pleasure to actually be the program chair was also because of those amazing track leads like every single one is a specialist in his or her community.

They start the fire in so many world class speakers, what world class keynotes that you are going to miss if you won't be at Index in San Francisco. When is that? February 20 to 22? Look at the keynotes, definitely going to be a really, most interestingly the schedule is way more interesting.

We have speakers from across the industry, so it's not an IBM show. We have Google, Amazon, Spotify, lightbend is there obviously with a talk. It is a community event centered around getting you started with everything that is of interest as of today for you completely. We are not solving that lifelong learning problem, don't get me wrong, but you will get a solid head start with index. It is not that expensive. I'm pretty sure if you follow me on Twitter you will find a discount code at some point. I'll make sure to tweet some out.

Everybody who is actually following at indexconf on Twitter has a good chance to sneak one or the other actual discount code.

Is there anything in particular you want me to talk about, Marc-Arthur?

>> MARC-ARTHUR PIERRE LOUIS: I think you have covered it all. We thank you for your contribution to this tech talk, and to your involvement in Index, and looking forward to meeting you there because I will be there and many on the team will be there.

>> MARKUS EISELE: Definitely. I'm looking forward to that too. Have a wonderful day wherever you are and maybe I'll be back at another tech talk some day.

>> MARC-ARTHUR PIERRE LOUIS: So, for as far as our tech talks are concerned, we have one for you on the 13th, and it is about SQL databases, doing access on a no SQL database which is a pretty interesting concept. That is February 13 upcoming.

We hope that you will be here to participate in this. Let me see if I can find it. If not, then it's going to be on spark on SQL on no SQL database, on February 13 next week. Thank you for your time. We are looking forward to seeing you again next week for another session of tech talks. Thank you and we will talk to you later. Bye-bye.

(end of call at 12:51 p.m. CST)

Services Provided By:
Caption First, Inc.
P.O. Box 3066

Monument, CO 80132
800-825-5234
www.captionfirst.com

***

***