EPISODE 1176

[INTRODUCTION]

[00:00:00] JM: Business intelligence is crucial for both internal and external applications at any company. There is a wide array of proprietary BI tools, and today there's an increasing number of options for open source business intelligence as well. One of them is Cube.js. Cube.js is an open source analytical API platform for building BI. Arteom and Pavel from Cube.js joined the show to talk about what they've built and their vision for the platform.

If you're interested in subscribing to the Software Daily podcast, you can get ad-free episodes if you pay $100 a month or $100% a year. It supports us, and you can subscribe by going to softwaredaily.com.

[INTERVIEW]

[00:00:46] JM: Guys, welcome to Software Engineering Daily.

[00:00:49] AK: Awesome, thank you for having us.

[00:00:51] JM: You both work on Cube.js, and this is something worth exploring from the analytical or visualization perspective. I'd like to get a bit of a lesson in the lineage of visualization, of web visualization. Could you tell me how web visualization products have traditionally matured over the last maybe decade or so?

[00:01:14] AK: Yeah, sure. I think that we are nowadays live in a world where software engineering is significantly easier than it was like 10 or 12 years ago, and same for you know visualization part. Right now we have a lot of great visualization charting libraries available, right? Back in the days it was pretty much only Dead Tree. And then we started to have a lot of great visualization products and like a lot of gradient charting libraries and a lot of you know integration with existing React or like Angular and other frameworks. So I think that right now

we have both really good developed ecosystem both for visualization and also for building things like reaper builders, pivot tables and general kind of like just dynamic dashboard thing. So I think that's really developed especially with the advantage of React, Angular and other like frontend, heavy frontend Jam style frameworks.

[00:02:14] JM: And in what ways do you see Cube building on top of the building blocks that came before it?

[00:02:20] AK: Yeah, I think that like, just what about [inaudible 00:02:25] in terms of as frontend becomes easier and easier and a Jamstack really becomes a sync. A lot of people they started to think about having multiple microservices on the backend and having a lot of frontend, kind of heavy frontend applications, probably built with like a React or Angular. And that's where Cube just comes in as like a missing piece being this API for analytics. So we see that Cube just fits really nice in terms of having a lot of microservices, and Cube just being one of them and kind of having a really heavy frontend application build, say, in React, which consumes as Cube.js analytics application. And then it can use any visualization, like to build any user analytics user experience that product people want.

[00:03:13] JM: Could you get a little bit more detail on the architecture of Cube.js?

[00:03:17] AK: Sure. You can think about Cube.js as an API middleware layer which sits between your databases and your frontend applications. So frontend application can query Cube.js. over the JSON API. And then Cube.js will generate and execute a sequel against to your databases. And also Cube.js kind of caches all the results and make the queries really fast and also manages things like permissions, authorization, identification. So it's usually kind of something between your database and frontend application. So that's why we call it API or middleware.

[00:03:54] JM: How does Cube.js fit into a stack with other common analytics tools?

[00:04:00] AK: Yeah. Since we don't really provide visualization and we've been visualization agnostic, sometimes we can be a back end for existing tools. We have a lot of cases where companies use tools like a Metabase or Grafana on top of the Cube.js where Cube.js acts as a backend just specifically to solve this data modeling problem or to solve their aggregation and caching problem and they just use Metabase as a frontend application.

[00:04:30] JM: So give a little bit more detail on what this kind of middleware – What problem it solves?

[00:04:35] AK: Yeah. I think that there are a bunch of different problems, but I would say the biggest one is low latency, is say imagine you're building a dashboard and you have a lot of filters on that dashboard, like 10 filters. Every filter have 50 values, 50 possible values and you have like a 15 charts on the dashboard. So you can think about potential number of permutations right here and different queries when you change values in the filters. And the question would be we have a really huge underlying dataset, and it could be multiple databases, and the data should be joined. How we can make it work really, really fast? And that's one of the biggest problem Cube.js solves is it creates what we call a pre-aggregation layer, which is sort of the caching layer. And it aggregates data from the source databases and to put it into aggregated storage. And then Cube.js queries this aggregated storage and it already supports all the possible permutation of different values, like all the filters and charts on the dashboard. That's why all the queries, they go and hit Cube.js, and Cube.js gets them from aggregated data, not from the source data.

And in the background, Cube.js is constantly refreshing this pre-aggregation layer just making sure that when you have a new data, the cache is kind of up-to-date. I think that would be the biggest problem. Also, it has like just obstruction, API obstruction, permission authentication, but performance is definitely the biggest problem.

[00:06:06] JM: So if I've got a bunch of different data sources and I want to potentially have visualization tools work on top of them, you're saying you do pre-aggregation, caching, generation of common SQL queries?

[00:06:21] AK: Yeah, that's correct.

[00:06:22] JM: How do you generate all that information ahead of time?

[00:06:29] AK: Yeah, it's a good question. So what you have is we have a concept of data schema. So the data schema is it's written in JavaScript. It's pretty much like a JSON object. In the data schema developers, they can describe the sort of the what kind of data they have and what kind of the relationship in the data they have. You can think about it like ORM, but for analytics. And based on this data we can create an optimal set of aggregation. Say, for example, in this data schema we have like 10 measures and 20 dimensions across multiple cubes. And then we can define a pre-aggregation to cover all the possible permutations of these measures and dimensions. That's why we can kind of can know ahead of time what's going on, because we operate on this kind of middleware layer with this defined data schema.

[00:07:26] JM: Now in some of these cases the datasets could be really large. Like if I have a Snowflake data warehouse, that could be really data intensive, right?

[00:07:34] AK: Yeah. In that case you really wanted to pre-aggregate things. So if you have like a lot of data – And when you build an application, imagine you're building an application for your customers, like end customers and you don't want to make them wait for five minutes even if you have like a lot of data under the hood in your Snowflake. You want to like basically provide a user experience when they click on something they got the data instantly back, right? That's why we built a pre-aggregation basically. The idea is you put a Cube.js on top of this and you apply the pre-aggregation and it goes and get all the source data from Snowflake. Aggregates it in a form of just basically the SQL table with a group by already applied. So all the columns are already aggregated and then it puts it into storage. And then when a query comes in from a frontend, it's not going to hit your original Snowflake database. It's going to hit this aggregation layer.

[00:08:33] JM: What are some of the engineering problems involved in building that out?

[00:08:37] AK: Yeah. It's really interesting. I mean like a lot, and Pavel can share some of them. But I would say one of the biggest interesting problems is building this storage layer, which is kind of we're using Rust to build it. We're using the Data Fusion, which is sort of the Apache Arrow implementation in Rust. And we also use a bunch of interesting tools like the RocksDB. But Pavel can share a little bit more about it and challenges. I think, Pavel, I will let you take it from here.

[00:09:06] PT: Yeah, sure. I mean, yeah, As Arteom mentioned, like the hardest part here is a Cube store. So currently it's based on Apache Arrow and Data Fusion. So these are query engines, like columnar query engines. And we use Parquet as underlying storage layer. And we use pretty the same way the Snowflake use like to approach processing a lot of data using micro partitioning. However, unlike Snowflake, we don't do it lazy. We do it like so every node has a warm-up cycle. So every node is always ready to query the data. And we like aiming for like 200 millisecond response for every query of any size. And it can be done using distributed like query processing. So using like this micro partitioning, so every partition is really small, like several megabytes. So we have like 100 of nodes that processes query in sub-second and providing sub-second response here. So like one of the biggest challenges is how we do this distributed processing, and especially like for problems like joins. So it's what we're selling right now.

[00:10:37] JM: Just so we avoid losing the lead of what we're actually talking about here. How does Cube actually improve the developer experience?

[00:10:49] AK: So imagine you wanted to build – You're kind of building a SaaS application and kind of big part of your application would be some sort of the analytics dashboard for your customers. You maybe wanted to build like a booking system for barber shops or like for any small business. And you wanted to show transactions, bookings and appointments and a lot of analytics around it. And to build that, you probably would – If you want to build it from scratch, it would involve building a lot of just blocks that you have to rebuild and rebuild over time. It's like pretty much like authorization, right? So people already did it multiple times, but it's hard to

do it right and it's a lot of like small problems and kind of caveats you should care about. So that's where Cube.js comes in. So it solves all the problems of SQL generation and data modeling and it just kind of drop in. You install a microservice, connect to your database. Define a data schema. And you should be good by kind of querying it from the frontend. So you don't have to worry about obstruction layer or API authentication and caching. So that's I think the biggest like developer improvements. It just kind of saves you time and like a lot of – Remove a lot of stress of building and maintaining this thing down the road.

[00:12:17] JM: Can you say more about a prototypical user of Cube.js?

[00:12:23] AK: Yeah we usually see like a lot of – I would say like two categories of the usage is the most common one is when company wants to build internal custom application. So imagine you're like a large company. You have a lot of employees and you're building some custom application for them to work with data probably to apply some custom workflows. That's kind of a common use case. And in that case you would see a team of internal engineers, like three, four people working on that application. In a lot of cases most of the work should be done on the frontend since kind of Cube.js is a backend infrastructure. You just need to configure it the right way and then you just consume the API and build your analytics to application.

And the next case would be more like a customer-facing. When you build an application and you want to expose some analytics to your customers. The composition of the team would be the same. We see a lot of you know like a frontend developers working with the Cube.js API. While more like a backend developers full stack developers are responsible of configuring and working on the data schema. Sometimes we have sort of the nice intersection with data engineering teams. When they like owns the data warehouse, they manage all the data inside the warehouse. They even use some great tools like DPT to transform the data. But then when it comes to like software engineers build the final application, they just give the connection to the, say, Snowflake. And then the rest is handled by software engineering team, which is a primarily user of the Cube.js.

[00:13:58] JM: So this is kind of an end-by-end problem, right? Because you have all these different data sources and you have Cube.js in the middle as middleware and you have all these different places you have to export your data to, like High Charts and Vue.js and React.js. How do you solve that end-by-end problem?

[00:14:15] AK: Yeah. Yeah. That's a very good question. I think that communities really helps us here. So initially when we released the Cube.js on GitHub in March 2019, we just pushed it out. We built I would say like probably like a two, three database drivers just to connect to Postgres, MySQL, and it probably was Presto or something like that. So we thought that it would be the most common. And then like communities, they started to develop different drivers. And by driver we mean like integration between Cube.js and database. We call that driver. And because like someone comes on a GitHub and say, "I wanted to use this with like a Vertica or Snowflake," and we didn't have any connection at that time and they started to build and contribute that back. So right now the majority of the drivers are contributed from the community and it's still like ongoing process. As we're just talking right now, people are building dynamic DB drivers or something like that.

And when we are on the frontend it was pretty much the same. Pavel and I, we kind of know React and we have been working with React for a while. So it was no brainer for us just to build some integration for react. Some nice kind of API wrappers to easily work with Cube.js API and React. But when we released that, a lot of people come to Repository and say, "Hey, what is about Angular and View?" And then people started to contribute it to Vue and Angular. So it's same here. So really rely on a community to help us to integrate with a lot of like tools on the frontend and databases.

[00:15:58] JM: How else has the community been helpful in building Cube.js?

[00:16:02] AK: It was instrumental. Like, really. I mean when we released the Cube.js, we put a Slack channel, kind of Slack instance just we thought that it would be good idea to have people hang out and Slack and give us feedback and we thought that in other successful open source projects it was like, "Okay, let's do just Slick and put out a Slack instance." And people

started to join it and people started to ask questions and started to tell us about their problems. And when we released Cube.js initially and then after the first release, we didn't even have any like road map or something. It was pretty much all community-driven. People were like either sending pull requests or giving us feedback. And we were just iterating on the feedback. It was pretty much you know like a talk to folks in Slack. Just try to process what changes needs to be done either in product or documentation and just then improve it. So I think that community really shaped the product. So I will say the community was the product manager for Cube.js, and it's still pretty much being a product manager.

[00:17:08] JM: You mentioned the storage layer that you built in Rust. Can you say more about that?

[00:17:12] AK: Yeah. So when we initially released the Cube.js, we didn't have our own storage and we relied on MySQL to start pre-aggregation. But there are a lot of problems with that approach because we quickly realized that it's just not a good like a tool for a job. And we started to think about different options to build that and we quickly realized that we want to have a columnar storage, which is going to be really optimized for ingestion and aggregated data and then querying because it's aggregated data. We really wanted to make the ingestion fast. That because when you want to build a pre-aggregation, like the ingestion times really makes a difference. Because if you have a low latency ingestion, you can refresh more often, right? And it means you can get updates to your data more often and you're going to end up with a fresh data. Versus if your injection is really slow and it kind of makes just – Your cache is not going to work.

So we really wanted to optimize for ingestion and also for querying obviously. And we started to look for different options and we understood that it has to be built like in low-level language just because in terms of the optimization, of all the possible optimization and low-level languages. And we ended up choosing Rust because it looks really promising. And I think it makes a difference in low-level programming. Pretty much like a C, C++ and Rust. You don't have any other options. So we chose Rust. We're really happy with that decision. And then we started to look for the architecture of how we want to do that. And we don't want to really to

write a query engine like from scratch. And Apache Arrow project was instrumental here. So we're happy to use it. And also as Pavel mentioned, we did some interesting kind of architecture decision specifically to optimize for working with aggregated data and optimized for joins across aggregated data.

[00:19:16] JM: Were you familiar with Rust going into it?

[00:19:19] AK: No. Pavel, he's doing the main development on the Cube storage right now. So he can share a little bit more, but he has a lot of experience with C and C++. And he was playing his Rust a little bit. I was playing with Rust a little bit, but no one really had a production level experience with Rust. And Pavel, do you want to follow up here?

[00:19:38] PT: Yeah, sure. Yeah I had over like four years of experience with Scala before like going into rust, and overall I have experienced like over like 10 years in Java. So it wasn't like a big leap for me. And I have a like native development experience, most of the time I spent with C++. And Rust comes here because of this kind of memory management thing. We looked outside Go lang, and the problem there is basically garbage collection and heap location. And we thought that it would be great idea to have this memory management thing. And it seems it works, yeah, for us. Yeah/

[00:20:31] JM: You also mentioned usage of Apache Arrow. Can you talk about that?

[00:20:36] AK: Yeah, sure. So Apache Arrow comes as no surprise here as well. If you know, like this Influx guys also announced that they're rewriting their core with Apache Arrow and Data Fusion as well. So Apache Arrow I believe becoming a standard for columnar data processing mostly because of it's basically aiming for SIMD optimizations versus single instruction multiple data. So it's much more optimized to process huge amount of columnar storage data formats, one of which is Parquet. And we use Parquet basically as a underlying storage layer for all the data written to Cube Store. I mean, currently it's no-brainer if you want to write columnar database. You would probably go with Apache Arrow here.

[00:21:41] JM: Have you paid much attention to the Apache Arrow community? I did a show about it a while ago, but I know the community has really matured since then.

[00:21:51] AK: Yeah, it's a great question. So right now we are at the early stage here. And Apache Arrow has matured over like time. I would think in like C++, Python languages. However, Rust is very like early there. So it doesn't have all the capabilities, for example, that C++ version has. But yeah, I think like Apache Arrow community has a great potential. And we are looking forward to contributing back to the code base and working with the community because we see a lot of like interesting projects starting up especially with this Rust Apache Arrow implementation.

[00:22:42] JM: So you've got this home built data store and then queries come in to the data store. Can you give me the lifecycle of a query?

[00:22:52] AK: Yeah, sure. So it's very like classical for any database storage. And again, we use for query processing data fusion. So we use like SQL Parser, which parses this SQL query into abstract syntax tree. So then it's planned. So we use data fusion query planner, logical query planner. At this phase we like intervene this process and introduce distributed query plan. So Data Fusion doesn't have distributed query plan as something that can be sent across nodes. We have it in Cube Store. And then when the query plan, logical query plan distributed across workers, worker builds its own physical query plan. This is also Data Fusion physical query plan and it's executed on a worker process. So results built on each worker and collected back using Apache Arrow serialization format. And basically send to a router node, which orchestrates all the querying process. So that's basically it in a nutshell.

[00:24:15] JM: How do you test the end-to-end data flow?

[00:24:19] AK: Yeah. So it's a great question. Like for Cube Store, we have basically integration test suite. So we started to write this test recently. And basically the idea is we're just spinning up a new like database instance, writing some data to it and testing wireless

queries under various circumstances. Different like configuration of cube store and different configuration of like querying, stuff like that, yeah.

[00:24:55] JM: So you guys are also a company. Can you explain what your company does?

[00:25:03] AK: Oh sure. We started the company around the Cube.js as a project. We called it a Cube Dev, because the cube is obviously taken. So we are kind of wanted to continue to develop the open source and support the community. But at the same time, to be able to do that, we just need to have some sort of the commercial offering to just basically fund the development open source. So we have been thinking about it in different ways to from a company to kind of – And start making some revenue out of it. And I think that idea was what if we like do not focus on support and services. Just build some sort of the ecosystem products around Cube.js while keeping Cube.js fully open source. It's under Apache 2.0 for the backend and MAT for the frontend and we wanted to keep it this way. And we don't wanted to do any open core and just release some part of the code as a different license. So the idea was to create a company around the product and just to start building an ecosystem and infrastructure tools around Cube.js which the revenue from this can support the development of Cube.js as well.

[00:26:24] JM: So have you built a hosted version yet?

[00:26:27] AK: We're building it. It's a lot of things to build, but we are doing a great progress here. So we have a few companies using already and it's been an interesting ride and a great ride. I think we're just around the last mile here, just polishing everything. Yeah, excited to launch it early next year probably.

[00:26:51] JM: And what are the difficult engineering problems of making a hosted version?

[00:26:56] AK: So it's all multi-tenants at first. It's all designed to be run like in a hybrid environment as well. It means that not only in our VPC in AWs, but potentially in a customer VPC. So we can just do it like multiple ways. The second way is – The second kind of a

problem is how we just make sure that we provide you really high reliable like uptime. And Cube.js has a lot of moving pieces, like APIs instances, worker instances, pre-aggregation layer, Cube Store itself, right? And also it's is free in memory caching and kind of queue synchronization. So it's a lot of things that we need to like provision and manage. But interesting thing is that we're building a development, like a workflow, and let developers to not only put a production code and run it, but also to be able to run sort of the development mode while the development. Like kind of sort of the live version while you do changes in schema, the Cube Cloud gives you a live preview of the API. And then once you feel that it's ready, you can just commit and push into production. Pretty much development workflow we're all used to, and we wanted to incorporate that into Cube Cloud. And it means we have to be able to really spun up these development servers. And once the code base changes, schema changes, we kind of do hard changes and hot reload. So I think that was one of the most complicated things we built in the Cube Cloud. And Pavel can kind of tell a little bit more about it probably if you want.

[00:28:33] PT: Yeah, sure. I think one of the most interesting engineering problems here is also application performance monitoring. And the reason for it is I think we are first time ever trying to solve this in SQL. So like Cube.js monitoring tool is very similar to what like Data Dock or New Relic provides. However we provide Cube.js specific metrics around like querying around pre-aggregating the data around database, query orchestration, stuff like that. So the challenge here is no one did it in SQL in past. And we use like food doc principle here. So we're using Cube.js to provide this APM analytics for Cube Cloud users. So that's interesting change here, yeah.

[00:29:29] JM: Tell me more about some of the infrastructure decisions you're making.

[00:29:33] PT: Yeah, I think I can take this one. So in terms of infrastructure – So currently we use Google Cloud to host main application cluster. And as Arteom mentioned, it's fully multi-talent. So [inaudible 00:29:50] has its own server no, it has its own database. So no one intersects in terms of data or like data access. And for every like major cloud provider we have uh like special clusters which sits in its own region. So we can host like deployments around

the globe using this approach. So basically every region of AWS, Google Cloud and Azure has its own cluster we support. So when people select where they want to deploy, they basically select the cluster where they want like the deployment. And we will do all the heavy lifting for them.

[00:30:42] JM: How big do you think the market is for a pre-aggregation middleware solution like this?

[00:30:49] AK: We're thinking of our market generally as embedded analytics market first. So it's a huge market with a lot of existing like companies and Bi vendors offering sort of the embedded BI and like revenue share of companies like a Tableau. Like about 15% comes from the embedded analytics and the market in general is really big. And what we see is companies with all the recent shifts in the technology, companies leaning towards building internally rather than using old Iframes. That's we saw initially with the problem while we started to work in Cube.js. So I would say that the market is huge and the market is moving into modern stack direction, like favoring Jamstack applications, microservices and especially all these data lakes and data warehouses. So we believe that Cube.js has a really good position here to win this market down the road.

[00:31:58] JM: So let's say I plug my data sources into Cube.js, what happens instantly?

[00:32:04] AK: You would be able to quickly generate a schema on top of this based on your data tables. And then you will have an API. And then say you have something like users tables in your database and you probably could have something like orders table in your database and then you would be able to ping an API and getting like number of orders grouped by users' ZIP code, something like that. And Cube.js will go and generate a SQL, generate a join, get the data back, aggregate it and send it back to you by the API. And on the frontend you can use any of our integration to build a chart like with React and charges, for example.

And since we all provide all kind of integration out of the box, you can do that under five minutes. And then you will be able to do something like, "Okay, I got users by ZIP codes to plot

on my map or like to plot as a bar chart." Then you would be able to apply filter, which is quite easy. Just update a JSON object. And again, Cube.js will generate all the SQL. So you don't have to go and update the SQL. It's all generated.

[00:33:19] JM: Is there GraphQL support for the querying layer?

[00:33:23] AK: No, not yet. It's an interesting question. Like a lot of people ask that and a lot of – We see a lot of users use any like GraphQL backhands together with Cube.js. So for example, someone want to use like Hasura. We use Cube.js and they would query analytical queries with Cube.js. And then if they wanted to, say, for example, save a state of the dashboard, they would use Hasura to save the state of the dashboard over to GraphQL. So in that case they use kind of Cube.js roll up and GraphQL for OLTP queries. So we see that happening all the time right now specifically using like GraphQL as like a spec, as a language spec for accessing Cube.js queries. We have been thinking about it. The problem is I believe that GraphQL is really designed to describe this sort of the transactions, like operations you wanted to do on transaction data, like OLTP style, right? And it's not really have a lot of good like tools and a way to express and analytics queries or lab queries. So I think that probably an like going down the road, we may want to adopt GraphQL over JSON at some point in the future. But right now the format is JSON just because it feels that it's just a better like a way to express the query.

[00:34:54] JM: What other features would you like to build into Cube.js?

[00:34:58] AK: Yeah. Well, one of the kind of biggest features we planning to support is real-time backen. So right now we're very good with either transactional sort of the databases, PostgreS, MySQL style. Well, folks, say, connect group just directly to say PostgreS replica. All with big data kind of a backend like a Presta, Snowflake or BigQuery. What is the missing piece here is real-time and streaming data. And by that I mean one is kind of event streaming, like a Kafka. We plan to support Kafka with ksqlDB quite soon. And the next step would be like a change data capture, like streams. So that would be like the next frontier for us. Once we

have all of this, we would be able to provide features like enriching data, like in reaching your real-time data with your historical data or vice versa.

[00:36:02] JM: Have you seen any applications of Cube.js that have surprised you?

[00:36:07] AK: Yeah. I've seen that. We usually didn't like initially expect that everyone will going to build pretty much like either customer facing sort of the embedded analytics or some custom analytics application. But then we started to see an interesting headless use cases where, users, they don't build UIs in the frontend and they just use Cube.js for automation of alerting system. And we've seen several interesting use cases like in automation where like Cube.js API is constantly requested for like changes or updates. And then based on this data, they perform some actions like updates in a factory line or something. So that was interesting. We also recently started to see a lot of use cases around machine learning. So we initially didn't like really envision that field yet, but we started to see a lot of usage from Python and people building like using Pandas, something like this with Cube.js. So we started kind of – We're still not sure about like moving into that direction like a machine learning, but that's something definitely interesting. And we see a lot of usage here recently.

[00:37:28] JM: Is there anything interesting to say about shuttling data between the client and the server in these data heavy embedded analytics applications?

[00:37:38] AK: Do you mind to elaborate what do you mean by shadowing data?

[00:37:41] JM: Well, I imagine if you have like tons and tons of data in this pre-aggregation server and you need to get that data to the frontend, there can be a lot of a lot of data. Like there's a lot of bandwidth that needs to be occupied.

[00:37:56] AK: Yeah. And, Pavel, feel free to follow up here. But I would say that since we work mostly with aggregated data, right? So it's not like someone would expect to move a lot of source data, like raw data between frontend and a Cube.js API. So we usually talk more about kind of small queries with aggregated data. We encourage everyone to – If it's possible to

move to WebSockets, which is kind of saves time on a kind of transmitting data. And we prefer our users to use a lot of small queries instead of a few larger ones. But, Pavel, feel free to kind of follow up here as well if you want to add something.

[00:38:43] PT: Yeah, sure. I mean if you take a look at this problem from edge computing perspective, then like Cube.js approach is that serving nodes which are like basically services that serve data to clients should be on edges. And also Cube Store layer should be on the edges as well. So users shouldn't access any raw data at any time. So they use like this aggregated storage layer to access the data. This way there will be always like best possible latency here in terms of accessing the data as opposed to accessing like one central database that have all the data.

[00:39:31] JM: How does Cube.js compare to other kinds of embedded analytics toolsets like – Oh, I don't know. I guess nothing comes to mind. But are there any main competitors in mind?

[00:39:46] AK: Yeah, we don't have really like a main competitor. Like the biggest competitor would be do it yourself without Cube.js. I mean like is there a category of Iframe-based tools but they're not really competitor? It's just a different local level. It kind of could like solve the same problem, but it's mostly used in like a POC or to build something quick. And then if you want to really move that into production, you either can tolerate the Iframe experience. Of if you cannot tolerate it, you have to build your own. And there we go, it's either do it yourself or Cube.js.

[00:40:27] JM: What's the importance of Cube.js being open source?

[00:40:31] AK: Well, we're software engineers, and if I wanted to use a developer tool and tool like a Cube.js, I would like to have it open search to be honest. And I believe that it feels right just for this category of software to be open source and it makes a lot of difference. You can see the source code and not only see the source, but you can contribute. And I think it's not only about the code being just open source. It's more like about how community can be built

around this and how people perceive it. So I think, to be honest, my opinion would be that all developer tools are all like infrastructure tools that should be at some sort of open source, at least part of that. So it should be open source. And then it feel like a modern way to just build software tools.

[00:41:24] JM: Well, is there anything else you guys want to add about Cube.js?

[00:41:28] AK: No, I think it was really great. Thank you for asking all these questions. It's been a pleasure.

[00:41:33] JM: Absolutely!

[00:41:35] AK: Yeah, awesome.

[00:41:36] JM: Thank you guys.

[00:41:37] AK: Awesome, thank you.

[00:41:38] PT: Thank you.

[END]