EPISODE 1032

[INTRODUCTION]

[00:00:00] JM: Datomic is a database system based on an append-only record keeping system. Datamic users can query the complete history of the database, and Datomic has ACID transactional support. The data within Datomic is stored is an underlying database system such as Cassandra of Postgres. The database is written in Clojure and was coauthored by the creator of Clojure, Rich Hicke. Datomic has a unique architecture with a component called a peer, which gets embedded in an application backend. A peer stores a subset of the database data in-memory in this application backend improving the latency of database queries that hit this caching layer.

Marshall Thompson works at Cognitect, the company that supports and sells the Datomic database. Marshall joins the show to talk about the architecture of Datomic, its applications and the life of a query against the database.

As always, we're looking for show ideas. If you have a great set of topics that you're interested in hearing more about, send me an email, jeff@softwareengineeringdaily.com. We're looking for interesting guests, great topics to cover in the world of software and exciting conference talks or podcasts that you've heard. You can also tweet at us @software_daily.

[SPONSOR MESSAGE]

[00:01:26] JM: DigitalOcean makes infrastructure simple. I continue to use DigitalOcean because of the low friction and attention to user experience. DigitalOcean has kept the experience simple and I can spin up a server in less than a minute and get high quality performance for a low price. For an application that needs to scale, DigitalOcean has CPU optimized droplets, memory optimized droplets, managed databases, managed Kubernetes and many more products. DigitalOcean has the flexibility to choose the right instance for the right workload and he could mix-and-match different configurations of CPU and RAM.

If you get stuck, DigitalOcean has thousands of high-quality tutorials, responsive Q&A forums and a customer team who treats customers respectfully. DigitalOcean lets developers focus on what they are building. Visit do.co/sedaily and receive $100 in credit over 60 days. That $100 can be put towards hosting or infrastructure and that includes managed databases, a managed Kubernetes service and more.

If you want to get started with Kubernetes, DigitalOcean is a great place to go. You can use your $100 to start building your distributed system and you can get that $100 in credit for free at do.co/sedaily.

Thank you to DigitalOcean for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:03:03] JM: Marshall Thompson, welcome to Software Engineering Daily.

[00:03:05] MT: Hey! Thanks, Jeff.

[00:03:08] JM: You work at Cognitec, which helps make the Datomic database. Datomic was started in 2012. Describe the principles behind the Datomic database.

[00:03:19] MT: Sure. Yeah, Datomic has been around in one form or another. Now in a couple of forms since 2012, as you said, and it was developed by – Well, architected by Rich Hickey, who is the CTO of Cognitec and also the author of the Clojure programming language. Unsurprisingly, a lot of the principles behind Datomic itself are very akin to those behind Clojure as a language and as a philosophy. Those include things like immutability all the way up and down, simplicity at the core and sort of opinionated perspective that rich and the Clojure ecosystem have sort of contributed to software in general I think are well-reflected in Datomic as a product and as a database.

[00:04:01] JM: The classic model for applications that interact with a database backend is the client server model. How does Datomic differ from the traditional client/server model?

[00:04:12] MT: Yeah, that's a really interesting question, and I think there's sort of an original easy answer and sort of a more subtle answer that follows from it. The answer three or four years ago would have been, "Well, Datomic isn't a client/server database because it uses something that we called the peer library instead of a client library. The key difference there being that calls to do things in the database, whether it's run some query or do some ordering or whatever the work is that the database would traditionally do actually occur in your own application code in the JVM where your app is running. That's why we called it a peer instead of a client.

In the years since with the release of Datomic Cloud, which I'm guessing we'll talk about in a little while, the lines have blurred a little bit and that we actually do have the ability to use Datomic in a more client/servery sort of sense. There is the Datomic client API and the Datomic Cloud database acts a little more like a server where you send a request over the wire and the database some work and then it turns around and it sends you an answer. But, of course, as we release that, a lot of customers said, "Hey, this whole cloud thing is cool, but I really miss that in my app processability locality with the data, being able to control where this work happens."

In the time since then, we've actually released some additional features for Datomic Cloud including what we call ions, which is where you can deploy your code into the running JVM that's up on the Datomic Cloud system to bring back some of that sort of database in your local process nature that was really popular with the peer API.

[00:05:47] JM: You mentioned this peer API, this concept of peering. This is pretty core to how Datomic has historically worked. Can you explain what that term peer means?

[00:05:59] MT: Sure. Absolutely. The Datomic database is a decentralized database and the idea here is that instead of having all the work the database does, whether it's storage or writing things to the database or reading things from the database all in one place, which is sort of the traditional model that you' see in a RDBMS. Datomic is distributed.

We have a process called the transactor that's responsible for doing the write work. It's what actually persist data that you have transacted to the system. We also have in the example that you're discussing, Datomic peers. Peers are actually any JVM that's running the Datomic API as

a JVM dependency, and those peers are able to read the database but not through that transactor instance that I talked about. I actually read directly from the disk storage, the persistence layer. Datomic also sort of outsources the persistence to other options, and that makes it sort of modular. The peer library is able to read those persistence layers directly into your JVM and it includes the facilities for local caching and using additional caches if appropriate.

But what that means is that you essentially get unlimited horizontal read scalability by spinning up more of your app servers. Any JVM that is running this Datomic peer library is considered a Datomic peer and it can read the entire contents of that Datomic contents from storage, perform all the work of query and it does all that in-memory on your process. Then whenever your application says, "Hey, I need to write something to the database." It issues a transaction and that's sent then over the wire to the transactor instance that I mentioned, which is actually the bit that's responsible for persisting that information.

[00:07:45] JM: As you mentioned, Datomic uses an underlying storage service and this means that there's an actual underlying database to a Datomic instance, like Postgres, or Cassandra, or DynamoDB. Why do I need another database that sits under my Datomic database?

[00:08:06] MT: Sure. The way Datomic uses storage – And you're correct, many of the storages that are available are themselves databases, is largely a key value sort of usage. Datomic writes chunks of data to it as sort of opaque blobs that are indexed with UUID. The actual semantic access to the data in Datomic is only through the Datomic APIs.

When we talk about using Datomic, the schema that you can use, the way that your data is laid out, the hierarchical categorization of your entities and the relationships between those entities, all of that semantic information is only available through the Datomic API itself, because Datomic is responsible for making as something that your system can consume.

The decision to make a pluggable storage actually was made to make it sort of more portable and more available in many places. Some customers with Datomic on-prem say, "Well, I have to run my own data center and our DevOps people say you can use anything you want as long as it's Oracle." Others say, "Hey, I'm 100% on this cloud thing and this DynamoDB sounds great. I

want to use that." The ability to sort of outsource or step away one layer from the persistence layer itself allows you to deploy Datomic in a lot of different scenarios that may or may not be available to everyone.

[00:09:25] JM: Datomic is written in Clojure. Can you give a brief overview of the Clojure language and why it's useful for a database like Datomic to be written in Clojure?

[00:09:34] MT: Sure. Clojure is a list that runs on JVM. I would say about 2005 or 2006, and I'm not the Clojure historian here. You should talk to Alex Miller, Alex or Rich. Rich Hickey who has spent most of his career working on large distributed transactional systems felt that there might be a better way of doing this sort of work. He took a sabbatical and authored the Clojure language as his approach to the way that he wanted to build software. Clojure values a lot of things fundamentally differently than some other languages. I already mentioned immutability. Clojure is fully immutable language. By default, everything is an immutable data structure. Clojure is very data-oriented. It's functional programming language. A lot of these principles are the things that Rich felt strongly were better tools for building large distributed complex systems in a way that was both simpler and easier to maintain.

All of that influences also the design of Datomic. Datomic comes from the same set of principles that things should be modular, things should be functional. Data is what's important, right? Not necessarily objects are abstractions, but the actual data and the transformations of that data. Also, the system should be designed in a simple way to make them future proof in the sense that they remain easy and straightforward to maintain and extend in the future.

[00:11:04] JM: A Datomic database stores a collection of facts in the underlying storage system, and most databases, you think of them as storing documents, like a NoSQL database, or in the case of a relational database, you think of a rows and columns. What is a fact? Why am I building a database out of facts?

[00:11:29] MT: Right. Another way of saying that is that Datomic is actually a topple store in similar ways to, for instance, RDF. So the relational data framework as a 3-tuple that's been around for a very long time, and the idea behind that is it's an entity, an attribute and a value. You can say lots of – Almost everything you want to say about information and data modeling as

represented by that entity attribute value pairing. So you can say things like Jeff is an entity, right? What does Jeff do? Jeff hosts. What does Jeff host? He hosts this podcast. So that's an entity is Jeff. The attribute is what you do, and the podcast is the value of the thing that you do that with.

However, there are some definite shortcomings with just an EAV tuple model, and one of those is the nature of time. This is unsurprisingly anyone who uses most traditional databases. There are lots of different approaches that people have taken, but often it's very cumbersome to sort of model when did this happen in my database? What was the state yesterday? Is it possible for me to rewind what my database looked like yesterday or last Tuesday or last Thursday versus what it looks like right now?

That's where the fourth element of the Datomic tuple. Datomic is actually a five-tuple. It's entity, attribute, value, then transaction, and operation. Transaction represents the fact that Datomic remembers the transaction time. So that's essentially the transactor machine time that every fact has been added to the system.

What this means is that you can do what I just suggested. You can very simply ask Datomic, "Hey, what was the state of my database last week at 2 PM?" and you get a value of the database back that represents exactly what that looked like, which, it turns out, is extremely powerful for debugging, for auditability, which is one of the places where we see a lot of traction as far as our customers being interested in Datomic and its applications. The final element of that tuple that I mentioned is the operation. So that's whether or not you're adding this fact saying this is true now, or you're retracting a fact. Marshall likes pizza. I'm going to say that's true as of right now. Then two hours from now, Marshall no longer likes pizza. So I can retract that fact. This speaks now to the fact that I mentioned that Datomic is an immutable database. We can't just take things away. If we want to remember what the state of the world was last Tuesday, I can't go in and delete the datum that said, "Hey –" The fact that it said, "Hey, Marshall like pizza." I have to now assert that I no longer like pizza. That's what the last element of that tuple is for.

It turns out that this five-tuple model is very powerful and flexible way of modeling data that lets you do things, as you suggested, like row modeling. A row equivalent in Datomic is give me all

the facts about a given entity. That's sort of like a row query. So find all the facts that have the same E. On the other hand, you can do things like say, "Hey, tell me all the facts about the same A." That's a little more like a column-like query of the data. You can also have datums about other entities. There's a type in Datomic called a reference, and this allows you to model things like hierarchical and graph structures because you can traverse from one entity to another through a relationship type of fact.

[00:14:44] JM: How does the sequence of facts, how does that compare to a database replication log? I know like Postgres, for example, has this log of transactions that you can roll back to or roll forward at any point so you can replay the database to any particular time. Is a fact, the fact store for Datomic, is that service similar purpose?

[00:15:16] MT: That's an interesting question. The answer is sort of it depends, yes and no, depending on how you like to look at it. Datomic does have a transaction log that is very much analogous to sort of a traditional SQL transaction log and that is an ordered series of every transaction that has ever happened. We didn't mention it yet, but Datomic is a fully ACIT transactional database. Every time you send a transaction, that is either fully committed or fully not committed and it's done durably in an isolated fashion.

There is a law log in that sense of the word where you can say, "Hey, what was the previous transaction? What was the one, three transactions before that?" However, that data structure is not inherently particularly powerful for asking questions about unless your question is specifically what just happened or what happened at three happenings ago. What you need then are more powerful indexes for asking questions like the ones that I just suggested, "Hey, I want to know about the entity Jeff." I don't care when it was established that Jeff host this podcast, or I don't care when it was established that Marshall likes pizza. I just need to know that fact.

Datomic has what we call multiple spanning indexes. I talked about the datum has these five parts; EAVT and op. Those facts, those datums are actually stored in Datomic more than once, and the reason for that is that they're stored in many different indexes that are ordered in different fashions. We have an index that you E-leading, right? When I ask questions about the entity Marshall or the entity Jeff, we know we can look in this index that's sorted by E's and will

find all the Jeff datums together. Similarly, there's one that's A-leading. When we ask about you know hosts or what people like, we can go find all the datums about that in the same place in that other index.

Sort of a long answer to your short question is, yes, Datomic has that transactional log, but it also has additional spanning indexes that provide the ability to ask more sophisticated questions in an efficient way than you would normally expect from just a sort of appended transactional log.

[SPONSOR MESSAGE]

[00:17:28] JM: Today's episode is sponsored by DataDog, a cloud scale monitoring service that provides comprehensive visibility into cloud, hybrid and multi-cloud environments with over 250 integrations. DataDog unifies your metrics, your logs and your distributed request traces in one platform so that you can investigate and troubleshoot issues across every layer of your stack. Use DataDog's rich customizable dashboards and algorithmic alert to ensure redundancy across multi-cloud deployments and monitor cloud migrations in real-time. Start a free trial today and DataDog will send you a T-shirt. You can visit softwareengineering.com/datadog for more details. That's softwareengineeringdaily.com/datadog and you will get a free T-shirt for trying out DataDog. Thanks to DataDog for being a sponsor of Software Engineering Daily.

[INTERVIEW CONTINUED]

[00:18:28] JM: I'd like to know more about these facts. Can you tell me like how does a fact get created? So if I do a write to my Datomic database, what kind of fact does that result in and what's my interface into that information?

[00:18:48] MT: Sure. Facts in Datomic, or datums, as we call them, are literally just these tuples that I mentioned. The shape of the datums that you can put in your database is defined to some degree by the schema that you've installed. It turns out that like in Clojure, Datomic treats everything it possibly can as data, and that includes the schema.

There are some requirements, for instance, your scheme attribute. We've we been talking about the attribute what kind of food I like. That may be an attribute that I've installed in my system and it may have the type string, right? I've said anytime I'm going to talk about things I like, those things are going to be string. But the actual requirements for the schema are fairly minimal and, again, they're just data.

The way that you put the schema in and also the way that you add any data to the system is through the transact API. Datomic has Clojure APIs for Datomic Cloud, and Datomic on-prem we actually have Clojure and Java APIs and you basically pass the transact function, eithers lists or maps. For those of you are familiar with Clojure, this is not surprising. Edn, being the extensible date notation is the language that Clojure speaks as it were, and it is similarly language or the data format that Datomic speaks. You can write those new datums, the new facts you want to add in a couple of ways. The most basic way is a list that says, "Hey, add," and then you give it an entity and attribute and a value, and the transaction time gets added automatically by the transacting system. It's literally just a vector of four things, add your E, A and V.

We also have the convenience sort of wrapping up something called the entity maps. It's often much more useful to be able to say a bunch of things about a given entity at once. I want to put in, "Okay, I want to create a new entity called Marshall, and it has a name, and it has a Social Security number, and a phone number, and an address, and maybe some of the things it likes. I want to wrap all of that in sort of some convenient data structure." In Clojure, the way we do that is you put it in the map and each of the attributes will become an attribute of that particular entity. So everything in that map is about that one entity, Marshall, that's being created as part of this transaction.

Then the transactor does the work of actually splitting that up into a set of individual datums where your address, maybe one datum with a string, and your phone number maybe one data, one another datum that's a string and your Social Security number maybe – I don't know, a long, or whatever. The sort of specific data types of those things are.

What's interesting is if you actually look at the return value of that call to the transact function, what you'll actually get back is a set of the expanded individual datum's that were added. If

you're sort of curious, “Hey, what is this entity map actually turn into when it’s put into Datomic?” The return value of that call actually includes a list of all the specific individual datums that were created as part of that transaction.

[00:21:44] JM: I’d like to revisit this architectural piece that you touched on called the transactor. Can you explain what the transactor is?

[00:21:51] MT: Sure, absolutely. This is now were speaking specifically about the Datomic on-prem product, because the Datomic Cloud product has a slightly different architecture. The transactor – I mentioned early on that Datomic is a modular distributed system and that there is a part that does the write. There's a part that does the work of query and the reads, and then there's the storage that are all sort of separate instances or separate systems. The transactor is the one that's responsible for doing those writes. That is a process that you're running presumably in your data center or on any AWS instance, and its primary job and, really, it's only job is making sure to provide the ACID fully isolated transactions for that system. It accepts calls to the transact API that we just discussed from peers and it persists those datums to storage.

Secondarily, I mentioned earlier that there are these other indexes that provide efficient query of different shapes and sizes. The transactor is responsible for periodically folding all the new datums that you’ve put in into those persistent disk indexes. That's a job we call indexing, which is a pretty standard thing that most relational database systems do. You amortize the cost of sort of folding in all that novelty into your persistent indexes by accumulating some of it over a period of time and then periodically writing that and then incorporating into the overall persistent index.

[00:23:18] JM: What are the conditions – I mean, would I ever need to scale up my transactor like under heavy load? Do I need to have multiple instances of a transactor is it completely at singleton concept?

[00:23:32] MT: Right. Again, speaking specifically of Datomic on-prem, the transactor is – Datomic as a single writer system. This is one of the ways and it’s probably the mode most important way that Datomic insures total ACID compliance, right? There's no sense of you can tune up or down the ACID level. There is no isolation factor like you'd see in some SQL

databases. With Datomic, you get all ACID all the time. One of the ways that that's ensured, is it only one thread? Only one process is ever responsible for writing to that transaction log that we discussed, and that lives in the transactor.

The question of scaling is an interesting one. Real quick, as an aside, you often do run two transactors, because Datomic provides high-availability. You can run a second transactor. It actually sits in standby and it monitors the heartbeat that the primary transactor writes. If you have a system failure or a network hiccup or something that takes that initial transactor or that active transactor out of commission, then you get a failover occurrence without any loss of transactionality or data loss or anything like that. You do get the high-availability side.

But as far as scaling, you wouldn't scale by increasing the number of instances. You can certainly scale the size of that transactor machine and the JVM that runs on it. One of the unsurprisingly frequent questions we get is, “Well, yeah. But I want to scale bigger than that.” But one of the places where Datomic differs is I’ve sort of already discussed, from a lot of more traditional database systems, is that you don't have to serve all of the read load through that transactor instance. It's only job is managing the incoming stream of novel writes.

What that means for most systems is that you no longer have to worry about doing things like you would in some sort of more relational traditional systems where you have to build all these read replicas and manage how you route traffic to one versus another and all these sort of stuff because your peers are where the actual read work is happening.

What that means is that the overall amount of work that's being done by the transactor is often less than people expect when they're coming from sort of a more traditional RDBMS system. Having that be the single point where writes go through is very frequently not a big issue as far as scale is concerned.

[00:25:50] JM: What's the process for me to set up a Datomic database? Is it any different than setting up a typical SQL database or a NoSQL database like Mongo?

[00:26:01] MT: I would say this is a place where Datomic Cloud, which is a newer product than Datomic on-prem particularly shines. Datomic on-prem, which is largely what we’ve been

discussing, is distributed as a JAR file. So you sign up for a license and you can download the distribution and then it's up to you to sort of put it up on a server and start it. We have some facilities for helping manage that in AWS and there are some community solutions for doing that on sort of other cloud providers or on-prem, but there's a little bit of ops involved, right? Granted, there is also sort of a local dev mode if you just want to try it out with local disk persistence. That's quite straightforward.

On the other hand, Datomic Cloud, which is a product that we launched about two years ago now, uses all the same semantic guarantees as Datomic. All of these stuff about how datums work and these ACID promises and a lot of this that we've already talked about are 100% applicable to Datomic Cloud, but the difference is that Datomic Cloud is specifically an AWS marketplace product. What this means is that launching a Datomic Cloud database is essentially a one-click experience. You go to the Datomic Cloud marketplace listing. You agree to the subscription and you click launch. Then 5 to 8 minutes later, you have a Datomic instance running in your AWS account that you could connect to either from your laptop via an SSH bastion or from a client instance running in your AWS account.

[00:27:29] JM: And if I'm setting up a Datomic database, what about defining my schema? Is there anything unusual I need to do when I'm defining the database schema?

[00:27:40] MT: That's it really good question. I think unusual is an interesting word for discussing that, I think. As we already discussed, Datomic schema is just data. The exercise that I tend to like to encourage people to do is sit down and think about how you would model your domain once you sort of know the very basics that are needed to write the schema, right? You need to know how do I define an attribute? What are the value types I can choose, et cetera, and sort of draw picture, right? Make a table of your various entity types and what the relationships between them look like, and then essentially if you sort of draw – I'm not advocating going all the way to UML necessarily, but sort of like an entity relation diagram, if you will.

Every attribute in each one of your entities essentially becomes a schema element. So then you write those up. You write them into an edn file and you can transact that whole thing to your Datomic system and you're ready to go. Now, inevitably, you'll get it wrong the first time,

because that's part of the learning experience, but also that's how data modeling often go is we learn new issues, new business questions come up. We realize that maybe we didn't model something the way we actually thought we should have. One of the advantages that Datomic has there is that, again, because schema is just data, you can update, you can add schema more or less anytime you want.

We've talked about the Marshall entity. Six weeks from now I realize, “Oh, man! I really wish that I had tracked the shoe size of my people entities, because I've decided to pivot and now I’m a shoe company.” Well, instead of having to sort of do a giant ETL or figure out some other way of handling that, you just say, “Okay. Well, now I have – Now I want to add a new schema element to my database. We’re going to call it shoe size and it's can be an integer,” and you transact that schema and you're done, and you can start adding shoe size values to any existing entities or new entities sort of as you see fit.

[00:29:33] JM: We’ve touched a little bit on this model of run time that Datomic uses, the peer model, where you have a peer component that gets embedded in the application, and the peer component can store some data in-memory near the actual application. Does this mean like if I am running my web app, let’s say I’m running softwaredaily.com and I make some query for a set of users, am I fetching the data from my local application, from that application that’s sitting in my browser?

[00:30:20] MT: Generally, I would say not in your browser necessarily, but certainly, possibly from your web server. That caches in the JVM. Your web application – This is a little bit of a hairsplitting thing, but you wouldn't run the peer library in a frontend app like JavaScript, for instance. It definitely is a JVM resident library. Otherwise, yes. I mean, what you said is largely true, and what's interesting is for users who have relatively small working datasets – So if the actual set of data that your app cares about is fairly small, a surprisingly large percentage of that may end up in cache in your application server instances.

This means that writing queries against it is essentially as fast as you can get things back and forth from your web server. I think one of the things that makes that even better is that Datomic is able to use Memcached. We have the ability to stand up – If you stand up a Memcached cluster, you can configure both your peers and your transactors to use that, and we've definitely

seen that some customers who have modestly sized databases can stand up Memcached clusters that are approximately the size of their database. If that's the case, then almost all queries you ever serve are coming from a Memcached request by your web server as supposed to a fetch from disk in the way that you would have sort of in a more traditional RDB round-trip sort of system.

[00:31:51] JM: I understand. Okay. This is what I was confused by. In an situation where, for example, I might be using Mongo and I'm making a query from my web application frontend. I hit my web application backend and the web application backend hits the database service and gets out of the database, which is on some different server. In the Datomic world, I'm going to have a local in-memory system that is managing sub-subset of my entire Datomic database that is on the same server as my application backend.

Now, what I'm curious about is if I'm storing some subset of my entire database on the application backend, how much of the database am I storing? How big is the subset of my entire database that's going to get stored on my web application backend?

[00:32:58] MT: Sure. That is a configurable parameter. It's called the object cache. It's specifically the bit of cache that we're talking about in this case. The size of that by default if you're using the peer library is set to half of your JVM heap. But that is fully configurable. One of the neat things about the way that works – Yes, your sort of description of that is exactly right. Your app is in the database in the sense that not only is it on the same server, it's not even a sort of out of process call. This is all happening in the same JVM where you have the peer grants you access you to a local copy, local value of the database is the way we like to think about it. You're right. Some subset of that is live in-memory in your Java system.

Now, obviously, the object cache is cashing Java objects, which are fairly fat in-memory relative to other possible ways of caching things. It's probably unlikely that for any sort of sizable system your entire database is going to fit in object cache. But then that's where you moved to then having Memcached, as I mentioned, as sort of the next backstop, where a fetch from local memory is what? Nano seconds, I guess, for object. You're talking less than 10 milliseconds for a fetch from Memcached versus tens to 100 for a storage git potentially.

[00:34:26] JM: So if we were talking about a database read from my application frontend or some operations going to result in a database read that's coming from an application frontend and it's going to hit the Datomic cache, or I don't know if you call it a cache, I guess the peer embedded application. How would the latency there compare to if I was using just – Let's say, I was using MongoDB or a Postgres and I just had like a Redis cache or a Memcached cache that I'm managing myself?

[00:35:06] MT: Sure. It's obviously very hard to say, because it's going to depend a lot on what's in your object cache versus what isn't and also potentially how much work has to be done, and in both of those cases. I think independent of whether you're talking about Datomic or sort of a more traditional relational database system. There is both the latency of, "Hey, I know exactly where this bit of information I need is on disk I have to grab it," versus, "I have to do some munging and I have to search some key space and I have to do a join." But if we try to sort of equalize all that out as best we can, the general experience that our customers have is that when you've set up a Datomic system with sizable caches and things are tuned well, it's very, very fast, a lot of these requests.

The interesting thing I think about this is we mentioned horizontal rescaling. I would say that a really cool approach that you can take when using Datomic is if you have a fairly complex system that has many different features and sort of frontend or more user-facing use cases, you can spin up a peer, your individual application server, for each one of those use cases. Because the object cache in that instance is tied to the behavior and the specific questions that instance has been asking and serving, then the result of that is that each one of those could have its own sort of finely tuned cache for the job it does. You may have one web server that's responsible for sort of your user login account management stuff, but you may have another one that's responsible for some other aspect of your system and you may have an entirely separate web server that's dedicated to serving back in more analytics queries for your quants.

The individual object caches on each one of those are going to differ based on the bits of data that each one of them have been responsible for answering questions about. Then if you sort of route your requests appropriately so that more analytics queries hit that same analytics-focused instance, that's a higher likelihood that the sort of data they're going to be asking for is already hot in those caches.

[SPONSOR MESSAGE]

[00:37:22] JM: Over the last few months, I've started hearing about Retool. Every business needs internal tools, but if we're being honest, I don't know of many engineers who really enjoy building internal tools. It can be hard to get engineering resources to build back-office applications and it's definitely hard to get engineers excited about maintaining those back-office applications. Companies like a Doordash, and Brex, and Amazon use Retool to build custom internal tools faster.

The idea is that internal tools mostly look the same. They're made out of tables, and dropdowns, and buttons, and text inputs. Retool gives you a drag-and-drop interface so engineers can build these internal UIs in hours, not days, and they can spend more time building features that customers will see. Retool connects to any database and API. For example, if you are pulling data from Postgres, you just write a SQL query. You drag a table on to the canvas.

If you want to try out Retool, you can go to retool.com/sedaily. That's R-E-T-O-O-L.com/sedaily, and you can even host Retool on-premise if you want to keep it ultra-secure. I've heard a lot of good things about Retool from engineers who I respect. So check it out at retool.com/sedaily.

[INTERVIEW CONTINUED]

[00:38:59] JM: I'd like to revisit the question of a read and then we can get to a write as well to get a picture of the database architecture as a whole. Let's say I am making a read of just a single record from my web application frontend, can you just walk me through how that read propagates through the Datomic system? Maybe you could give examples of the data being in that application cache, the peer component, as well as what happens when the data is not in that peer component.

[00:39:41] MT: Similar to the transact API that we talked about before, there are a couple of read functioned APIs for Datomic. Probably the most common one you'll see initially is the query API. Datomic uses DataLog as its query language. Let's say you sent a Datalog query to your

application server in one form or another. Now in that system you've invoked the D/Q API. What happens is Datomic, the peer library, will compile that query as necessary and say, “Okay, I know what I need to look for.” Depending on the complexity of that query, that may be – It may involve one or more of those indexes that we talked about before that it knows how to most efficiently find things that match the parameters that you’ve asked for.

It will then say, “Okay, I know what the overall tree of these indexes look like.” Datomic stores all of its datums in a shallow tree and it'll say, “I need to find the segment where this particular set of Datums foo is at.” That segment is going to be identified by a UUID and maybe I already have it in my local cache.” If that's the case, then it can read it from the local object cache. It can feed it back into the query engine that's responsible for doing the joins and whatever, and then obviously as additional bits of data are return to it as appropriate, it can do those joins and return your answer.

If, on the other hand, it says, “Okay, I know I need this particular bit of data foo from this index,” it's not in my object cache. I’m going to go look in Memcached. It says, “Okay. Well, let's presume it’s not in Memcached this time.” The peer library does not go to the transactor. It goes directly to the storage backend. Remember, we said you can use many different pluggable storages, whether they’d be a SQL store, or a Postgres, or a DynamoDB, Cassandra. The peer itself accesses those storages directly and says, Hey, give me the segment with the ID foo that I know these bits of data that I'm interested in is located in. Then if you have Memcached configured, it will sort of the same time that is done that read from storage, it will write that segment into the Memcached instance as well as bringing it in to your peer application, caching it there in the local object cache and then it will do sort of everything I talked about before, which is it will look in there. It will find the specific datums that it knew it was looking for, pass them up to the query engine for either return or inclusion in additional joins.

[00:42:08] JM: How does a write contrast with the read path of Datomic?

[00:42:12] MT: Right. If your application says, “Hey, I need to transact this data. Marshall shoe size is 10.” You've called the D transact API that we’ve mentioned before, and what the peer does then is it says, “Hey, this needs to be persisted. That's the transactor’s job. It sends that information, that transaction itself, across the wire to that transact or instance that we talked

about where it's put in the write queue. You can either do this synchronously or asynchronously. There are APIs for both. Let's assume you're doing synchronously. The peer will then be waiting for a return value from that transactor. The transactor will grab that transaction data and it will persist it to disk. Before it ever acknowledges that any write has been achieved that you're guaranteed that that write has been persisted to that transaction log that we talked about before.

What that means is if you peer, if your client instance has said, "Hey, I got an acknowledged write," and then immediately all the power in the US goes out. When that power comes back on and you turn your database back on, your guaranteed that that data will still be there. The transactor will take that transaction data. It will do any work that it needs to do. So Datomic allows you to specify that things might be unique, for instance. You can have unique values and unique identities. That transactor may have to do a little bit of work to say, "Okay, you've just said that you want to create an entity called Marshall," and the entity name is something you said can only be a unique value." It may have to make sure that there's not already a Marshall in the database, that you're not doing something that's sort of not permitted by your uniqueness constraints. Once it's made sure that everything is good there, then it persists that to the disk and it returns an acknowledgment to the caller that the transaction has been persisted.

[00:43:57] JM: Are there any conditions that can lead to data consistency issues with Datomic, like inconsistent writes or reads?

[00:44:05] MT: Right. No. Datomic is from the very beginning, as I sort of briefly alluded to before, intended and designed to be a fully ACID system that doesn't allow you to end up with staleness or inconsistent views – Datomic considers consistency the primary goal of the database. If there is a situation on which Datomic has to choose between consistency and availability, it will choose consistency.

[00:44:35] JM: What about – What are the conditions that can lead to latency in that availability scenario or some kind of inability to serve a request due to low availability?

[00:44:50] MT: Sure. Speaking from the write side specifically, that connection to storage, right? Obviously, this is all distributed computing. Things happen. If the transactor can't reach storage, it does have significant retry built-in. But after a certain period of time, which is configurable, at

least in Datomic on-prem, we call the heartbeat. If the transactor is unable to write to storage, it will say, “That's it. I can't persist things. This is a consistency violation. I'm going to shut down.”

If you get a heartbeat failure, Datomic’s transactor will terminate. That's where that HA failover that we talked about earlier comes in. As far as sort of on the support side of things, that's one of the predominant number one causes of sort of Datomic systems going out or going down is, “Everything was great,” and then it went down because storage was configured or someone turned off storage not realizing that it was related to Datomic or that sort of thing. That's the predominant one.

Obviously, again, being a distributed system, just general networking sort of messages could not be passed between peers and transactors or transactor in storage can lead to some amount of latency in a system, but usually either the built-in or some client level retry can handle most cases of that that we see in the wild.

[00:46:05] JM: How does database indexing work in Datomic? Does the database make any indexes automatically, and what happens when a user wants to define their own index?

[00:46:17] MT: Right. That's a great question, and it comes right after the sort of path that we talked about with respect to writing datums. As we mentioned earlier, Datomic is a multiple spanning index database. So it does in fact maintain numerous indexes and those are differed by the sort of the elements of the tuples in Datomic, right? Those datums are sorted either by AEVT, or VAET, or EAVT, and Datomic automatically manages and updates those indexes in real-time effectively. The way that it actually does that is it advertises the cost of putting data into those indexes over time.

As you accumulate a certain amount of novelty, by default that's like 32 megs of new data, Datomic will kickoff and indexing job where the transactor says, “Okay, I need to fold this stuff into the persistent indexes on disk.” It does all that work and it writes the new segments to storage and/or Memcached if configured, and then it notifies all the peers, “Hey, there's a new index route. You can just start using that now.”

Prior to that indexing job, all of that novelty was held in what we call the memory index. It's not like you can't see that data from your queries until the indexing job happens, right? It's always visible and it's always there. The question is whether Datomic is fetching that from the persistent written on disk storage index or whether it's fetching it from a memory index that it folds in as part of the business of answering queries.

You'd also asked about sort of individual user specified indexes Datomic on-prem included one optional index, which was the AVET index, where you could specify for certain attributes, "I want that index turned on." Datomic Cloud actually, by default, turns on all of them. The reason for that is that there was initially the idea that might've been a slightly more expensive index to maintain for Datomic. But after a couple of years of sort of having customers use it in fury, it turned out that it was not a problem. Our recommendation now is actually to just generally run with that on all the time, because it makes life better for queries that can take advantage of it. It turns out not to be overly expensive for Datomic to do that.

As far as custom indexes in the sense that you would sort of think of them with SQL or a DynamoDB or something like that, until recently, there was sort of no facility for that. But what's interesting is we've recently released a feature called tuples. This is, before this, all V positions in datums, so the value. They were all scaler types. We've now released the ability to put a variable length tuple of data in that the V position. You can have a V position that is three things, a string, a string, a string, or a long, and int, a string, whatever.

It turns out that one of the – One or the reasons that – One of the strong reasons for doing that was that lets you build compound or composite uniqueness keys. But it also turns out that you can essentially leverage that feature to automatically get custom indexes on any pair or three or four set of individual values you care to. Because Datomic is already building all those indexes we already about, anywhere that that V is, whether it's the AVET index or the EAV index, it will sort those tuples lexicographically, the first, second, third element.

What that means is if you put two things that you care about asking questions about together into a tuple in the V position, you've automatically created sections of your overall index that acts like secondary indexes would in sort of a more traditional relational database. You can write your query to specifically say, "Hey, I'm asking about the combination of the person's name and

address.” Go to the section of that index via a very efficient tree look up that talks about all the person address value associations as supposed to find me all the people, find all their addresses, then find all their names.

[00:50:22] JM: Datomic has the ability to query the history of the database built-in. It has an automatic versioning history. Can you describe the conditions under which people might want to query the history of the database?

[00:50:40] MT: Sure. Yes, that fourth element of the datum that we talked about is the T. That's the transaction ID, which is associated with a time of every transaction that goes into the Datomic system. What that means is that every fact has a timestamp effectively. You can, as you said, ask what – “I want to run this query about people and their addresses and I want to run it right now, but I also want to run it and I want to see exactly what it would've returned to me last Tuesday at 2 PM.” Similarly, I want to ask about everything that's changed since last Tuesday at 2 PM and now. Those are slightly subtle differences, but they are different things you're asking.

Datomic provides out-of-the-box the ability to do all of these things by applying a filter to the database that we call the time filter. You can say, “As of you.” You can say, “Show me the database as of last Tuesday at 2 PM,” or you can say, “Show me all the things that happened since time foo.” There are a number of very interesting applications for this. I think one of the places where people have gained a ton of advantage from this certainly is in debugging, right?

A user reported a problem and now it's fine. Everything is working fine now and I don't understand why. Must've been something weird with the state. I guess we'll jock it up to I’m never get to see that again. With Datomic you can say, “Well, the user reported the problem on Tuesday. 17 people have made schema changes and updated things and whatever since then.” But I can say to Datomic, “Hey, what exactly was the state of the database when this user reported this anomalous behavior?” Then you can look in your code and say, “Oh, look. It's because X, Y, and Z,” which was actually mitigated by some changes that have happened since then, for instance.

Another place where we see a lot of traction with the built-in notion of time in Datomic is in regulated industries. We think sort of financial services, healthcare, insurance, these kinds of places where there's at least a business level, if not frequently a sort of legal level requirement for maintaining a history of what has happened. I need to know exactly when so-and-so's account balance changed, or I need to know exactly when the status of this patient was changed from A to B.

Obviously, there are plenty of ways that you can sort of layer on top of any existing application some of those features, but the idea with Datomic is you don't have to, and there isn't the chance that, “Oh, we wrote a bug into that system, so all of our history tracking just doesn't work.” The database is responsible for maintaining exact record of how all the data in there has changed.

[00:53:20] JM: Datomic uses a query language called Datalog. Can you explain what Datalog is and how it's different from a more traditional query language like SQL?

[00:53:29] MT: Sure. Datalog, formally, is a subset of prologue, which is a logic programming language. One of the most significant differences between Datalog and Prolog, is that Datalog is guaranteed to terminate, which is obviously a desirable feature of a query language that you’re using for a database. Formally, mathematically speaking, Datalog is equivalent to the relational algebra plus recursion, which means from a sort of formal methods perspective, you can do anything you could do with SQL with Data log and possibly even more if you consider that you can do recursive things.

Datalog uses a pattern matching approach to writing queries. It's very logic-oriented in the way that, I would argue, a good query language should be. In general, a lot of people sort of approach it and they say, “This is kind of weird. I don't really know what I think about this. I know SQL already.” But after a day or so of playing with Datalog, it really starts to shine in the sense that because, again, Datomic uses what we call the universal relation, this five tuple datum, and the data log query language equivalently uses this relation, your queries look like your datums. You're writing a query that includes a set of patterns that match what the data in the database, and furthermore, the data in your data model represent. What that means is that it's often very straightforward to translate your query from language from, “I want to know X,” into a set of

Datalog clauses that are very logically consistent with both your data model and the question you're trying to ask.

[00:55:07] JM: What are some of the downsides of using Datomic? What are the things that people struggle with?

[00:55:12] MT: I think there's certainly a bit of a learning curve, especially for people who have a strong long time commitment to some other technology or technologies. As you just pointed out, it's not a SQL system. It uses Datalog. Some of the ideas behind Datomic may seem a little different. This idea that your database never forgets. You don't overwrite data, like, "Wait, how do I write my program if I can't tell it to delete things?" There's a bit of an adjustment, I think, in terms of the way that you would approach problem-solving.

In general, people who come from the Clojure ecosystem have a very easy time with it because, of course, it reflects many of the same values and ethos as Clojure itself. But I think that we see a lot of people in a lot of different technologies and industries and backgrounds really gravitating more towards data-oriented systems that rely on the principles of functional programming and isolation and all these principles that Datomic really resonates with.

One of my biggest suggestions to people is just give it a shot. I mean, spend a little bit of time. I understand that trying new sort of unfamiliar things can often be a little bit of a challenge. But more often than not, people come away quite pleased that they have spent the time and sort of learned the interesting differences that there are to learn from Datomic.

[00:56:32] JM: Okay, final question. Datomic has been around since 2012. How has the database evolved since then and what are the plans for the future?

[00:56:42] MT: That's a great question. I joined Cognitect in 2014 when Datomic had been out for, yeah, about two years, a little less than two years, and I've been formally on the Datomic team since about mid-2015, I think. In that time, we've seen a huge number of changes. The first and possibly most obvious big changes that we've release Datomic Cloud. This is an entirely new codebase. It's entirely new database. We didn't talk as much about it. It, again,

provides many of the same semantic guarantees and functional capabilities as Datomic on-prem, but with a significantly different set of architectural decisions.

For instance, there's no longer a transactor. It uses a client API but has the ability to do things in the cloud. It includes as well actually a deployment model that we call Datomic ions, where you can actually ship your code up to the running database and sort of use it as an application deployment platform. There are many other aspects of Datomic Cloud that are sort of unique and very interesting, which I love to talk about sometime, but we certainly have spent a lot of time working on that aspect of it.

Additionally, since I've joined, many large features like tuples have come out, which have really extended and expanded the capabilities of Datomic as a database system both in terms of its actual semantic functionality and also performance and these sort of other important aspects.

Another really exciting thing that we released recently is something we call analytic support. This actually speaks a little bit to your previous question. Analytic support is a set of features that we've included that allow you to actually connect more traditional SQL-based tools to a Datomic system. Using analytic support, you can do things like connect Tableau, or connect PowerBI, or Apache Superset to your Datomic system and run your business analytics tools against it without having to do separate ETL into whatever data stores they prefer to have. It provides essentially a subset of a SQL interface to Datomic that's tailored specifically for sort of the analytical side of things. That's available both in Datomic on-prem and in Datomic Cloud.

I've also been fortunate enough at the company to work a lot with our customers. I spend a lot of my time talking to either existing or potential customers about what sorts of problems they're trying to solve. Why they’re potentially considering Datomic? What other solutions they've looked at? What kind of problems they're having with Datomic or without Datomic? I've been able to find a really interesting groups of people who have communicated and conversed with who’ve all come to this technology from many different places, many different industries, because of a lot of the underlying systemic value propositions of both Clojure and Datomic as an ecosystem and as an approach to writing software carefully and systematically.

I think that going forward, that's one of the places where Datomic shines now, always has shined, and I certainly think will continue to shine, is that it holds these sort of principles around simplicity and immutability and the way that we should be thinking about treating our distributed systems very carefully.

A lot of the other features are wonderful and exciting, but many times some of those features get in the way of they're polished on top of needing to remember that you have to be very conscientious when you're talking about the database of your system, that the foundations are really strong. That what I would say is one of the greatest strengths of both Datomic on-prem and Datomic Cloud and I certainly think that that will absolutely hold true in the future.

I guess the second part of your question is where are we going from here? We're consistently developing additional features, additional tooling. I'm personally involved with a lot of our customers on the side of sort of how can we make it easier to use Datomic. Even if it's not a surly feature side of things, like what are the aspects where Datomic is either difficult to learn or you found something unexpected or surprising about it or our tooling and documentation isn't as good as it should be. I'm heavily focused on improving all of those things as much as possible.

I also think that both Datomic on-prem and Datomic Cloud continue to grow in terms of our user-base and the number of companies, and to some degree, even the size of companies that are using it. I would expect that in the next few years we'll see quite a bit more community-sourced sort of content and I'm excited to – I always love to hear what people are doing with Datomic that isn't something I would've thought of. Yeah, I think that the biggest place is that I'm excited to look forward to are those things that a lot of our customers are coming up with that are really cool applications of the technology that we've put together.

[01:01:30] JM: Marshall, thanks for coming on the show. It's been great talking.

[01:01:33] MT: Thanks, Jeff.

[END OF INTERVIEW]

[01:01:43] JM: Gauge and Taiko are open source testing tools by ThoughtWorks to reliably test modern web applications. Gauge is a test automation tool that makes it simple and easy to express tests in the language of your users. Gauge supports specifications in markdown, and these reusable specifications simplify code, which makes refactoring easier and less code means less time spent maintaining that code.

Taiko is a node library to automate the browser. It creates highly readable and maintainable JavaScript tests. Taiko has a simple API. It has smart selectors and implicit weights that all work together to make browser automation reliable. Together, Gauge and Taiko reduce the pain and increase the reliability of test automation.

Gauge and Taiko are free to use. You can head to gauge.org to know more. That's G-A-U-G-E.ORG to learn about Gauge and Taiko, the open source test automation tools from ThoughtWorks.

[END]