EPISODE 1164

[INTRODUCTION]

[00:00:00] JM: Data science requires data sets to be cataloged and indexed. The datasets are versioned and might be in CSV files in S3, or in a database, or in another data storage system. Splitgraph allows the user to query this data catalog like it's a Postgres database, routing queries to any dataset across your catalog. Miles Richardson is the founder of Splitgraph and he joins the show to talk about data cataloging and what he's building with Splitgraph.

[00:00:34] JM: Guys, welcome to the show.

[00:00:36] MR: Thanks for having us.

[00:00:37] AI: Thanks. Thanks. Thanks for having us, yeah.

[00:00:40] JM: Start off with a simple question. What are you building with Splitgraph?

[00:00:43] MR: So basically how you can think of it right now is what looks to you like a Postgres database with 40,000 different tables in it that come from all these datasets all over the web, right? So what we're calling it right now is a data delivery network and there are basically two kinds of types of data on the website right now that you can find in the catalog and you can query with your existing Postgres client. You can query live data, where we forward queries to the upstream source. In most cases, that's a data portal powered by Socrata, which runs basically the government data portals, or we can forward your query to a snapshot of data, which is basically like a version history of a dataset that you can build with the open source tools that are available from Splitgraph.

So we're calling it a data delivery network right now, and that's kind of the public use case. And then ultimately for businesses, we're looking to have internal-facing use cases for data cataloging and data governance. So that's kind of where we're going with this.

[00:01:48] JM: Why is the company called Splitgraph?

[00:01:50] AI: Because the domain name was available. Actually that's the only reason. So I think when miles originally pitched me the idea, he said, “Oh, I just tentatively snatched the name Splitgraph for this, because it has a graph in the name, which made it sound like slightly scientific.” And both like the dot-coms and the dot-IOs were available. But kind of later on we also managed to reverse engineer an explanation for why it's called a Splitgraph. So it kind of mathematically fits in the way that we merge data, and that basically a Splitgraph is something that you can partition into like a set where all nodes are connected together and another set where all nodes are sort of disparate. And the idea is that we kind of lets you join disparate datasets and combine them with your internal datasets to kind of add value essentially.

[00:02:36] JM: So give me a little bit more information about how the user interfaces with Splitgraph.

[00:02:41] MR: Sure. So there're kind of two primary components of Splitgraph at the moment, right? One of them is splitgraft.com, which is sort of the web interface. We like to think of that as the catalog. And then the other is what we're calling the data delivery network. This is basically a public Postgres endpoint or it's what looks like a Postgres endpoint as far as your clients are concerned, where you can connect to and query the data.

So on the web, we have sort of this point of discovery where you can search for datasets and find data that you like based on keywords or whatever. And then we also have the point of access where you're connecting your SQl client to and sending SQL queries. And because of that combination we can do a lot of really cool things. So when we say that we're going to be working on this sort of data governance feature, the big idea there is that by combining this sort of web interface with the point of access, the point discovery and the point of access, we can do things like have web controls for sharing who has access to which data, right? So you'll go on to splitcraft.com – So this is eventually not what we have now, but eventually you'll go onto splitcraft.com or your private company instance of Splitgraph and you'll connect your existing data sources through a no-code workflow in the website where you go in there, you put in your read-only credentials to your Postgres database, or your Snowflake database, or your Red Shift database or whatever. We introspect it. We figure out what's in there. We expose it as this sort of familiar namespace repository paradigm. And then you can query it through the point of access where using your Postgres client, or you can share various columns or tables with other users in your organization from the web.

So because we know exactly who you are when you get to the point of access, we know what ACLs and things we can apply to you when you go to actually query the data. So there's also the sort of open source component, which I think Artjoms might want to talk about.

[00:04:40] AI: Yeah, absolutely. So I guess that's kind of the other way you can interact with Splitgraph. So you cannot run it completely standalone on your local machine without sort of us getting involved. And you can sort of treat it as a hybrid of Git and Docker for data. So the idea being that it lets you take a dataset and package it up into a so-called data image that's versioned and that's kind of similar to a Docker image. So it's self-contained and you can sort of share it around with other instances.

So it's also decentralized, and you can spin up a Splitgraph instance on some other machine and push data around essentially. That's kind of I guess the way we started. And this splitgraph.com instance that Miles was talking about. That's really kind of like the GitHub for Git. So it's the publicly available Splitgraph peer that gives you extra perks essentially to all the datasets that you push out to it.

[00:05:33] JM: So as a user of Splitgraph, I see two opportunities. There's the data catalog within my own company, and then there's the idea that I'm accessing and democratizing data potentially to other companies. Is that correct?

[00:05:48] MR: Yeah, exactly. So we're really emphasizing the sort of delivery aspect of it at this point. So Artjoms' background is in finance. And when we first started working on this, like a big problem that we saw was basically, "Okay, if you're say a hedge fund and you're buying data, you've got all these data vendors, sometimes hundreds of them, and there's no standard way that the vendors are delivering the data to you." I mean, you might get something better than CSV, but it's going to be proprietary in some way, or it's going to lock you into some kind of ecosystem, or ultimately just going to be more friction for your relationship. And as a hedge fund, you need to have lots and lots of data vendors. So the more you have, the more friction it is and the more problematic it becomes, right?

So we really like the idea of basically simplifying it as much as possible for both the publisher and the consumer. So for the publisher, all you need to do is basically give us your read only credentials. And because we implement everything on top of this Postgres technology with FDWs, which are basically like a way of proxying SQL queries to upstream where you translate to whatever the upstream is, because we implement it like that, all the publisher needs to give us is their read-only credentials and then we can basically automatically introspect their data. Figure out how to package it up and apply automatic sort of snapshot and procedures to it and then give them distribution features, billing features, branding features. And then for the consumer, like someone like a hedge fund, for example, we can give them basically easier ways to manage the vendors, easier ways to catalog their data. And most importantly, a standard format to receive the data in and a way to query it with SQL and also using the open source SGR a way to build these sort of reproducible data images.

So I mean a big feature we have is this idea of being able to maintain your datasets, right? So say you have a data set with two inputs that come from two different upstream sources and update like every couple days or something. You want to keep your downstream dataset up-to-date, or else it's actually only useful when you make it. And you want to have the process of keeping that up-to-date be efficient and not have to redownload the thing if nothing's changed, right?

So we took a lot of inspiration from Docker when applying this sort of split file concept where you can sort of declaratively give a recipe of what SQL steps and what datasets you want to input to your ultimate data image that you're building. And then you can use that to make these maintainable data images. So when it comes time to update it, you just run SGR rebuild and it can use the local cache and know when the upstream datasets change so that it can efficiently rebuild your datasets in CI just like you would a Docker image to make sure that you're keeping your image up-to-date when Ubuntu changes, for example.

[00:08:34] JM: Give me a deeper explanation for the problems that Splitgraph solves for the user.

[00:08:39] AI: I guess from the point of view of publishing, as a data publisher, as a data producer, your core competency isn't really making sure that the data is kind of published in a bunch of formats and that your users can consume it in all the formats that they wish. Your competency is basically finding this data and kind of organizing it into some sort of an internal structure. So as an example, let's say you are a postcode publisher, a postcode data publisher. And way too often you have an issue where, say, as a publisher you have to make sure that your dataset is available in a CSV as a dump maybe, like as a SQL dump. And you have to maintain instructions on how to ingest it into various databases, because that's what your clients want, because they're not very sophisticated. And so they kind of want to be able to make sure that the dataset works with their own procedures. And what we think is the case is that most publishers aren't interested in doing that as much as maybe someone else handling these things. And we also think that the best way to kind of consume this data and best way to get this data is literally over a database, so over some sort of a connection that looks like a database connection.

And so what we essentially let publishers do is make this data available as a SQL endpoint so that consumers can – Well, first of all they can obviously just query directly over SQL. Also they can export it into any format they wish, because they can follow instructions on how to export this data from a generic Postgres database. And so that's kind of the problem that we want to solve into the publishing.

In terms of data discovery, I guess the problem there is as a large enterprise you probably have a bunch of different data silos. So this can be something like your own data warehouse, which is normally where most of these big gathering or the data under one roof projects end up. But you also have things like your production databases. You have some sort of a data lake maybe. You might have some ad hoc FTP and CSV files on an FTP server. You might be subscribed to some SaaS services. In that sort of area, what we would like to do is make it easy to publish this data on this internal kind of a Splitgraph catalog. And because we also let you access this data directly through the catalog, through this one SQL endpoint, this means we sort of merge together the idea of discovering this data in the catalog as well as also querying it. In that case, I respect you kind of want to make also this enterprise data governance easier.

[00:11:08] JM: What is the state of the art when it comes to data catalogs within companies?

[00:11:12] AI: I think there's a few. So it's a bit of a green field to be fair. There're a few companies around that that do it, and there's a few open source products. So there's things like Amundsen, which has been out of Lyft, I believe. Datahub, which is a spin out of LinkedIn. I think there's also Calibre. And I guess that's kind of there. The view we have is that while there is this kind of idea of a modern data stack forming, so you have some sort of standard now for ETL. You have a standard for transformations and you have a few pretty robust tools for doing BI as well as data warehousing. We think there is really no good catalog yet available. And these open source tools are kind of very similar to just being essentially like a web app with no extra features. And so what we would like to do is try and see if there's anything we can improve there in that area.

[00:12:05] MR: Just to add to that a little bit, I think the sort of key insight that we're bringing to the table is that a catalog to be useful and avoid things like drift where you have one representation of your data on the catalog and then one representation of the data in reality. In order to be useful, it should really be as integrated as possible with not only the data sources upstream, but also your users accessing the data. So that's where this idea of basically combining the point of discovery and the point of access becomes really important and enables us to add a lot of features to our cataloging that would be difficult to do without also having this abstraction on top of the point of access. So that's kind of the unique angle that we're taking. Sort of starting from this idea of data virtualization, if you will, and then adding features onto that.

[00:12:58] JM: Let's get into a little bit about the engineering behind Splitgraph. Can you give me an overview of the stack?

[00:13:04] AI: Oh, there's a lot of stuff in the stack. So I think a lot of Splitgraph is powered by Postgres. So that's the case in the open source Splitgraph. So the Splitgraph engine is essentially a Postgres instance, which we manage ourselves. Kind of we manage on your machine. So we add extensions to it to make this whole thing work. And a lot of code that is related to querying is actually us sort of using Python to manipulate the local Postgres instance. So when you're doing some sort of Git-like versioning operations with Splitgraph, it's really Splitgraph itself running Postgres like normal SQL table operations on the local Postgres instance. And then kind of around that idea we have some extensions on this Postgres instance to implement some more functionality. And those are sort of written in C.

On splitgraph.com it gets much more complicated. So a big part of splitgraph.com itself is this whole idea of a hosted Splitgraph where we essentially use the same Splitgraph engines that you would find on GitHub and we just put them in the cloud and we manage them ourselves and we orchestrate them properly to, say, make sure that when you're running a query on the DDN, we sort of put all the split graph images in place to make sure it's seamless for you. And that's the sort of the backend part.

There's also the sort of a classical web app backend where you have a database and you have like an API server. And in there we have some kind of weird things going on. So of course, we do use Postgres as well for our database backend. And some of our API code is written in python. But we also use API generation a lot. So essentially software that generates API gateways for you. One example is PostGraFile. So PostGraFile lets you – It's a tool that lets you point it to a database. It looks at the table schema. It looks at sort of all the stored procedures in that database, and it generates an API endpoint that you can then query with like any JavaScript client. So and the cool effect of that is that you avoid a lot of boilerplate. So instead of writing these redundant CRUD functions, you just point PostGraFile file at your database. And in a lot of cases, that's all you need. That's kind of going from the back to front. I guess, that's the architecture there.

On the frontend, Miles can talk more about that. But we mostly use React there. And there's a bunch of stuff around infrastructure and orchestration. There's a bit of lure. There's a bit of shell scripting. There is some Grafana and Elk, but that's more support. And yeah, Miles can talk about the frontend stack, I guess.

[00:15:41] MR: So just to be clear, like if you want to use Splitgraph, you don't even need to install like the open source SGR. You can just straight connect to the public endpoint and query it with your existing tools. But if, say, you decide you want to publish your own data to it or you want to make your own data images, then you might download this SGR package that Artjoms was talking about. That's the one that's basically the core of everything that Splitgraph Cloud or splitcraft.com is built out from this sort of SGR Splitgraph core that you can find on GitHub, which is like based on Postgres, written with Python, C and stuff.

And then yeah, for the splitgraph.com, like the sort of web interface catalog portion of it, oone thing that Artjoms didn't mention is we basically have a ring of PG bouncers that are the part that actually make it look like you're talking to a real Postgres database and then we script it with Lua and Python to do various authentication routines and query rewriting and filtering and basically making sure it goes to either the right upstream image or the right upstream data source.

And then, yeah, on the front end or sort of back plus frontend with the JavaScript stack, we use Next.js with PostGraFile file like Artjoms was mentioning. PostGraFile is awesome in terms of productivity. And then React, TypeScript on the frontend. It's pretty standard, the web app. We're basically Python/ TypeScript shop with a sort of sprinkling of various other languages for high-performance reasons in some places.

[00:17:09] JM: Tell me about the process by which the user integrates with Splitgraph. So at the moment you can use it sort of how you want, right? So if you want to just get public data and you want to just query public data, which includes things like Post GIS and geographic queries, then you can just take your existing Postgres client, whether that's like DB for DB Grip or whatever, and go to splitgraph.com or splitgraph.com/connect and you'll get the details that connect to the endpoint. And then from there you can just query all of the data on Splitgraph directly. You can join across datasets. You can join across versions of datasets. So that's sort of the minimum involvement if you don't want to download anything. You just want to use it and you just want to get the public data and query it. You just use your existing client.

Now, if you want to create data images, for example, or do various advanced features, then you would go to github.com/splitgraph and you go to the Splitgraph repository and you can PIP install Splitgraph and then you have this command line tool called SGR, which also includes a local Postgres engine that we can manage with the SGR tool plus Docker for you. And that Postgres engine has things like change tracking enabled. So you could, for example, say you want to like scrape weather data and keep aversion dataset or something, you would basically set up a scraper to scrape the data. Put it into your local Splitgraph instance, which is really a Postgres database with some extensions loaded. And then you would SGR commit. And because the change tracking picked up everything that you inserted into it, it'll know when you do SGR commit that that's what you're going to add to the data image. And then, basically, you start with a snapshot of whatever your initial data is. And then as you add these rows each day as the weather changes or whatever, each one of those is a commit and we only store the changes. And then you push out the changes incrementally.

And then once you push them out, they're part of the catalog. But in the sort of medium-term future, the way that we want you to interact with Splitgraph is say you've got your internally-deployed splitgraph.com instance, which will likely be available as something either like an AMI, an Amazon Marketplace, or something managed by us or various other options we're tossing around. But you've got your internal company, Splitgraph. You'd go there, you put in your read-only credentials for your warehouse and all from within the web without needing to download anything. So like no-code. But we introspect your data source that you just inputted the credentials for. We give you the option to sort of select various column mappings and preferences you might need. And then you just click ok and then it's added to the sort of internal DDN along with all your other data sources. And then you can join across them using all your existing tools.

So a big thing for us is really to just stay out of the way as much as possible and sort of we think that good tools like Git add a lot of value without you really noticing they're there until you need them. So we think it's kind of a similar thing with data where you don't need to know that you're interacting with a version dataset. As far as your tools are concerned, you're just talking to a regular Postgres table. But when it comes time to actually manage that and version it and push up some new commit, then you can start using Splitgraph and it'll be fully aware of everything you've been doing to the database.

[pip install**] AI:** I guess in terms of consuming data, Miles mentioned that you can connect to Splitgraph with any SQL client, but also there's a few other ways. So we offer a few ways to get data over HTTP. One example is we run on Postgrest, which is also an API generator, but this time we run it against Splitgraph images and that generates open API, compatible API endpoints for any image on Splitgraph. So you can easily generate some client code to interact with this image over HTTP.

And another one is we also let you run these SQL queries like over a REST API. And that lets you, for example, use Splitgraph data, say, in things like observable notebooks. So you can start up an observable notebook, run a REST request against the Splitgraph endpoint, get the date and plot it within like a matter of minutes.

[00:21:21] JM: Could you define the term a data image in the context of Splitgraph?

[00:21:25] AI: Sure, yeah. So data image is essentially a collection of tables. So it's a kind of a snapshot of a database at a certain point in time. So a Splitgraph repository is a collection of data images, so much like an image is a Git commit, and repository is a collection of commits.

Similarly, Splitgraph repository is a collection of images, which are themselves collections of tables.

[00:21:49] MR: Yeah, the term image comes from Docker. So you can build an image. You don't need to use split files to build an image. Like you can use this local Postgres database with change rack enabled and just insert rows and then sort of do periodic SGR commits in the same way that you can technically build a Docker image without a Docker file where you basically change whatever on the system and then you Docker commit it and then you're building an image. But we share a lot of the same semantics with Docker in terms of the way we do cache invalidation in between layers of the Docker image. So I mean, Artjoms can go more to detail there. But the long and short of it is basically that a data image is a reproducible snapshot most likely created using something like a split file, whether either manually or via some automated means.

[00:22:37] AI: Yeah. I mean in terms of caching, it's like Miles said, it's similar to Docker where let's say you have a sequence of instructions and each instruction is some SQL that you run against this repository. And so if for example you ask Splitgraph to run a sequence of instructions and then ask it to run it again, it'll know that the instructions are the same. And so it'll know that it has already computed sort of this transformation and it'll just give it back without having to recompute it. And that's really the caching thing here.

[00:23:09] MR: Yeah. Another big feature that this enables is sort of this idea of like a forking workflow, right? So if you think of this idea where maybe you start with one base data image and then two users want to make different changes to it, then what you would do with the analog of that on GitHub is you would fork the repository, right? Now if you implemented that naively with data, it would mean making a copy of the entire database for each and then you would basically have two duplicate copies of the base plus the changes that each person made. But with data images, it means that we only need to store the base once. And then because each layer is sort of basically like delta compressed representation of the changes, we only need to store what each user change and then one base. So it's sort of a primitive that allows us to implement much more advanced workflows on top of it.

[00:24:03] JM: Could you draw a comparison between split files and Docker files?

[00:24:08] AI: Yeah. Yeah. Yeah. Sure. So I guess if you look in a Docker file, a collection of commands. So it's a list of commands that you're running against this Docker image. And every command essentially runs against the result of a previous command. So for example, the first line in a Docker file is something like from Ubuntu latest and that says, “Okay, this is the first image that I'm running my commands against.” And then you say run apt-get update. And that means, “Okay, we're going to take this apt-get command and we're going to run this in the context of the previous image,” and so on and so forth.

So you kind of have this idea of building up an image with a bunch of layers where each layer is a command. And so when you build a Splitgraph image using a split file, we kind of borrow the same ideas. So in the beginning, you might have an empty image. You might say, “Okay, I'm building this thing from scratch, and I'm running some SQL queries to generate some data.” Or you can say, “I'm building this from a Splitgraph image.” So something existing, for example, from Splitgraph/2016_electionlatest, and that's your base image similarly to Docker. And then again next line would be something like run, or in our case we call this sequel. And that just means, “Okay, we run the SQL command against the previous layer and we record the results.”

One thing what we do different from Docker is we let you import data from other images. So for example, I can say, “Okay, this is me doing from the 2016 US selection image.” But then in the next layer I might say, “Oh, also from this census data repository, import the result of this query as this table.” And this kind of lets you merge a bunch of datasets together. So you'd be able to run joins between them.

And I guess another cool implication of that is in the case of Docker, a lot of your transformations are really self-contained, right? Because say you do apt-get update, and that really means, “Okay, the Docker interpreter has to go off to the internet and has to get, say, the latest packages.” Whereas in our case, if you say, “Okay, let's import this table from this dataset.” Because the data set is actually a snapshot, we know exactly the commands that you need to run to rebuild the dataset.

So essentially when you run a split file, we record all the commands that were used to build that image in the same image. And what that really means is that if you take an image, if you just look at it, you know exactly which datasets came into it and how to rebuild it. And this kind of lets you draw out these massive dependency trees between datasets and images.

[00:26:50] MR: Yeah. So that's kind of this data lineage use case that ties in well to data governance problem space as well. Oftentimes, it's important for analytics teams to know where their ultimate downstream artifacts came from and what the inputs to them were. So by being able to just take an image and then get its provenance, it's really powerful not only for being able to do efficient updates, but also know where your data came from. So we have some examples of this on the website where you can actually see the provenance and reconstruct the split file to make the same image even if you don't have the actual split file itself uploaded.

[00:27:28] JM: Are there limitations on the size of a dataset I could use successfully with Splitgraph? Or can I just go hog wild with all my data?

[00:27:37] AI: Yeah. Yeah, sure. Yeah. It's an interesting question. Partially because we're powered by Postgres, and that really means anything that works with Postgres. So any limitations of Postgres apply to Splitgraph. So obviously Postgres is pretty robust and pretty horizontally scalable. So that's kind of the upper bound. But beyond that, we also do some optimizations to kind of get Slipgraph working with larger datasets than normal. So as an example, for backend storage, we don't actually use Postgres tables. We use a columnar storage format called cstore-fdw. And so that's kind of a Postgres extension that uses the formats that's similar to Apache Parquet or ORC, and the idea is that it compresses better and it lets you run analytical queries much faster. Because essentially all the data is arranged column- wise rather than row-wise.

And so from that point of view, I guess, in terms of handling larger datasets, we also do a few other optimizations. So as an example, you can store part of a Splitgraph dataset in S3, so in object storage, and you can only have like a lightweight set of metadata on your local Splitgraph instance. And then when you query the table, we can inspect the query and we can see which sort of parts the dataset it requires and then download them kind of lazily in the background. And so for that point of view, it kind of helps you explore larger datasets much easier.

[00:29:06] JM: What's the difference between Splitgraph Core and Splitgraph Cloud?

[00:29:10] MR: So one thing that's really important – Well, there's multiple things that are really important to us. One of them is being developer first, right? We think it's really important that you should be able to try the product within 60 seconds and the full product and you should be able to do everything with the product. Splitgraph Core is the open source portion of it on GitHub, and you can do basically everything that we claim you can do with Splitgraph with Splitgraph Core, but it's going to be mostly useful in a either more individual or local context, right?

So you would use SGR for something like local development. And then splitgraph.com is actually basically like a super peer to SGR. So if you think of GitHub, that's like a Git peer. You can kind of think of the same thing with Splitgraph Cloud or splitgraph.com, right? But splitgraph.com can be deployed into different contexts. So at the moment it's deployed publicly. And so that's sort of our public instance of Splitgraph Cloud, in the sense that github.com is the public instance of GitHub, but you can also get enterprise licenses to have GitHub installed at your organization or your institution or whatever.

So we ultimately plan to do the same thing with Splitgraph, where in that context, Splitgraph Cloud will be your internal company deployment of Splitgraph. And then Splitgraph Core will be this portion that either you may download if you're doing things with it or you may just only need to interact with Splitgraph Cloud and you can be able to do all the configuration through the web. And maybe you're like an analytics engineer and all you need to do is write SQL queries against the data so you don't even worry about the Splitgraph Core portion. But yeah, basically Splitgraph Core is the open source component around which everything is built. We also call it SGR in various places, because that's the command line tool that makes up the bulk of it. And Splitgraph Cloud is the web catalog plus data delivery network, basically.

[00:31:07] JM: So again you have this term data delivery network. Can you define what that is and how much traction it has?

[00:31:13] MR: Yeah. So we've kind of coined this term, I guess. The idea of it comes from content delivery network. So it's meant to evoke that association of a CDN, right? What we like about this sort of idea of a CDN is, well one, it's very easy to configure, right? So with Cloudflare all you need to do to configure your website is change your DNS settings to point to the Cloudflare servers. We think similarly if we can make it easy for publishers to publish their data by just giving us read-only credentials to the database that presumably already exists if they're doing something like scraping data, because they need to be storing the data somewhere in some format. And as long as we can connect to that in a seamless way, we can reduce the problem for them to only inputting their read-only credentials.

So that sort of ease of configuration and then adding value-added services on top of that. Like once we have this live connection, we can start doing things like snapshotting, and replication, and versioning, and data governance, and firewalling. Being able to just put your read-only credentials in. And then from the web interface, configure how this data is distributed. That's kind of one aspect that we want to replicate from CDN to DDN. The other aspect of it is just this notion that the way that we distribute data on the web makes a lot of sense in terms of distributing data to a wide audience, right? So it's not like when you publish a new version of your website, you don't take a snapshot of it or a zip file and ship it off to every Cloudflare server in the world, right? They do it the opposite. They wait until a query comes in for your website and then they fetch a copy of it and then they might keep it around the cache and give it back to you or give it back to users that come to that same server in a short time period. But obviously, you're not just sending out and broadcasting the new copy of your website to every

Cloudflare node, because that would be unsustainable and you wouldn't be able to fit all the websites that Cloudflare is able to serve on its network.

So we took a lot of inspiration from that architecture, and working backwards from there is kind of how we got to this idea of data images, because it makes it a lot easier to cache and reuse data in small chunks when you have a lot of these semantics guaranteed already.

[00:33:32] JM: How does Splitgraph fit in with the other tools of the modern data ecosystem?

[00:33:36] MR: Yeah. So right now the primary use case of Splitgraph, how we're positioned, is basically just a place to get public data, right? So at the moment we don't really have or we're not positioned as a part of the stack, but where we're moving to, like, say, you want this internal deployment of Splitgraph, we're pitching our value initially as the data catalog.

So one big thing is we don't want you to throw out what you already have in your stack. We want to integrate with your existing stack. So basically we want to be something that you can deploy on top of it and instantly add value by cataloging the data that's already there in your data stack and giving you a way that your analyst can access it such that you can govern their access to all the upstream data and you can manage it really well.

But yeah, for us basically we see the catalog as sort of the initial value add. But ultimately, we would like to sort of replace this idea of needing to have a warehouse and ETL and ETL-ing things into the warehouse. I mean, this is sort of the long-term goal, but we think it makes a lot more sense to just sort of start from virtualization so that when you're kind of experimenting and figuring out what queries and what data you want to put into your data artifact, you don't need to talk to a data engineer to set up some ETL scripts to put your data into the warehouse first. You can just go directly to your downstream databases, which may not be live DBs. They might be either a replica of a live DB or an analytical DB. But basically if we make it easy to be in this experimentation phase, then once you decide that you do need some sort of replication, we can do it through Splitgraph and through the data virtualization. And then we can eventually get into this place where you have one solution replacing your warehouse and ETL. But that's kind of the long term. For now we don't want to replace anything. We just want to basically work on top of your data stack and support as many upstream data sources as we can. So initially, that's going to be analytics databases, like Snowflake, Red Shift, Postgres and we're really just going to emphasize these sort of upstream connectors and working with as many as those as possible so that we can easily deploy into a new organization and instantly add value just by connecting their existing data sources.

[00:35:52] AI: Yeah. I'll also add that currently there is a lot of tools in the modern data stack that just work with Splitgraph because we support the Postgres protocol. So as an example, on the ETL side, you can do something like run a single tap against the local Splitgraph instance. And because it's just Postgres, you can then take this result in table and run a commit on it, and there you go. You have a new Splitgraph image, which you can now upload anyway you want.

And in terms of transformation, similarly, you can run DBT on top of Splitgraph. And we even have some plugins for DBT that when you run it on Splitgraph and you reference Splitgraph dataset, we handle cloning and sort of making the dataset available locally. So in your DBT transformation, you can say, "Oh, I would like to create table as select star from "Splitgraph/2016 election", again. And that will behind-the-scenes go load this dataset for you and then put it onto the engine so that it's seamless for you.

And again, in terms of the final step of your data stack, which is BI tools. You can easily run something like Metabase on top of Splitgraph, because again, it speaks Postgres. So we actually internally do that to power our own analytics stack. So we run Metabase on top of a Splitgraph, and we make Splitgraph proxy to our a few data sources. So, say, Elasticsearch, for example, which we use for our event storage, and MySQL, which you use for web analytics. And so in front of that we have a Splitgraph instance, which federates queries across these things. And then on top of that we run Metabase to give us essentially a view of what's going on in the business through the data that we join across from all of these distinct silos.

[00:37:33] JM: What about the relationship between Splitgraph and data versioning tools like DVC or Pachyderm?

[00:37:40] AI: I guess in terms of these tools, yes. I say DVC is used to version data for ML projects and it uses a Git annex behind-the-scenes. So essentially it lets you add a reference to your Git repos, which references certain artifacts from your ML pipeline and then materializes them when you check out revision. And so the difference there is that we focus on tabular data. One thing that lets us do, things like delta compression. So if you check out a table, make some changes to it and commit it, then you that lets the store changes much cheaper. And also that means that because we're leveraging a database, that means we can actually query this data much faster. So it might be in some cases a better tool to use maybe in your ML projects. So you would use a Splitgraph image, like you would reference a Splitgraph image in your ML pipeline, for example. And then you would sort of check that out together with your Git revision.

In terms Pachyderm, I guess, I think we kind of have thought of an integration there where because Pachyderm lets you kind of snapshot the whole state of your data pipeline, and one of the stages of this pipeline can be running Postgres, you can feasibly try and run Splitgraph as a Pachyderm stage. So you would package up the whole Splitgraph engine as a Pachyderm kind of result, intermediate result. And then when you run this pipeline, you sort of unpack it and you run Splitgraph against it and then you pack it again.

[00:39:08] JM: The beautiful future of Splitgraph would be to allow a kind of community or social network of datasets. I think there's been some effort at having tried that in the past. What do you think are the barriers to getting to that bright future?

[00:39:23] MR: So I think one big issue that a lot of these attempts have had in the past is that they didn't put any kind of like standard format first. So they would either force you into some kind of proprietary data format, or you'd be limited to CSVs, or maybe you only work with Python pandas, for example. But we're really trying to build on open standards and open source and we really like Postgres. We really like the Postgres protocol. And we think that really by just removing as much friction as possible from both sides, so the data consumers and also the data producers, we can make it much easier to get to this sort of end state, right?

So I mentioned earlier, for the producers, just being able to only enter your read-only credentials makes it much easier to publish data. So there's actually this possibility that we could even unlock a sort of market segment of people that might be, for example, really good at scraping data for various use cases. And that's something that you find a lot in sort of the consulting/freelancing world, is clients who are looking for specific datasets that require kind of ad hoc scraping.

We have a good use case there, because you can write your scrapers, you can push your data into Postgres and then you can just connect that Postgres directly to Splitgraph. So by making it really easy for the publisher to do that, we kind of solve the bootstrapping problem in a way. We've also kind of mitigated that problem by leveraging these existing open data portals to start so that we can start with 40, 000 datasets. But then also by reducing friction on the consumer side for so that the people querying data can use their existing SQL client and their existing tools and integrate with their existing workflows without breaking anything or needing to change anything. We think those are kind of the most important aspects of it.

And then aside from that, it's kind of, I mean, obviously a big marketing exercise and just getting this community built. But yeah, definitely, that's a sort of utopia that a lot have imagined, and we'd like to get to at some point. I also think it'll inherently never be as big a thing as sort of GitHub is for code, just because workflows and data tend to be different. But it does seem like we're kind of missing this place that you can go to get access in a standard way using SQL to whatever dataset you want. And that's kind of where the original idea came from, which is very simple. Like what if there's this database you can connect to and all of the data in the world is available for you to query on it whether individually or, more interestingly, across different datasets. So just making the interface simpler and reducing friction and building on open standards and being developer first. I think those are really our key differentiators at this point.

[00:42:16] JM: Is there anything else you guys want to add about Splitgraph or the data ecosystem?

[00:42:21] MR: One thing we would say is if any of this sounds interesting to you, we definitely love to talk. We are looking for design partners at this point particularly for the private cloud. So if you want a catalog solution, for example, we're really keen on talking to people who are interested in that, and splitgraph.com, check it out.

[00:42:44] JM: Okay, guys. Well, thanks for coming on the show.

[00:42:45] MR: Thank you.

[END]