Episode Book

EPISODE #5909

Connecting Postgres to Google Workspace

PUBLISHED	12 October 2026 at 17:24 UTC
DURATION	28:29
LISTEN	myweirdprompts.com/episode/postgres-google-workspace-integration-99679/
HOSTS	Corn, Herman, Hilbert
SCRIPT MODEL	deepseek-flash
VOICES	chatterbox-regular (chatterbox-local)
PIPELINE	V5.4
BOOK GENERATED	19 October 2026 at 17:41 UTC

SYNOPSIS

Rita runs a two-person business on Google Workspace and wants a real business layer on top of it. This episode walks through the integration architecture: why PostgreSQL owns the relationships while Google stays authoritative for documents, messages and events; how service accounts and domain-wide delegation actually differ; and why service accounts can't own Drive files — a constraint that forks the entire folder design. Plus the real costs of Cloud Run plus Cloud SQL, and what Google's new Workspace API quotas mean for small firms.

TRANSCRIPT



Corn

Rita's got two people, six pieces of software, and one question that keeps coming back. She doesn't want to leave Google Workspace. She wants a business layer that actually talks to it.



Herman

And this is the part where most people either give up and go full no-code, or they build something so clever it collapses in eight months.



Corn

Right. Daniel's framing this as the integration episode. How does a PostgreSQL-backed app on Cloud Run actually connect to Gmail, Drive, Calendar, Docs, Sheets and Slides without duplicating everything and without throwing away what already works?



Herman

The whole thing hinges on one principle, and it's worth saying before we touch a single API.



Corn

PostgreSQL owns the business relationships. Google stays authoritative for the documents, the messages and the events it hosts.



Herman

That's the line. A single system of record does not mean copying every piece of information into one database.



Corn

And everything from folder provisioning to the Slides merge to the calendar reminders falls out of holding that line or breaking it.



Herman

So do we start with the architecture?



Corn

We have to. Everything else is downstream of the auth model, and the auth model has one constraint that basically decides the whole Drive design.



Herman

Let's frame the actual problem first, because it's a mismatch of shapes. Rita's business data is relational. Clients, projects, sites, milestones, invoices. That's tables with foreign keys and it wants to live in Postgres.



Corn

And her documents, her messages and her appointments live in Google's apps, where her colleague already works and where the designers already know the interface.



Herman

Neither side should be forced to become the other. You don't want to build a document editor, and you don't want to cram a client project hierarchy into a Drive folder tree as the source of truth.



Corn

So the backend holds the relationships and reaches into Google for the content.



Herman

And the way it reaches in comes down to two auth models. The first is a service account, which Google describes as an account that belongs to your application instead of to an individual end user.



Corn

Meaning it's the app's own identity.



Herman

For server-to-server calls where no human is involved. That's sometimes called two-legged OAuth. The backend wants to create a folder, it authenticates as itself, done.



Corn

And the second model?



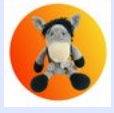
Herman

Domain-wide delegation. If the app needs to act on a user's actual data, a Workspace super-admin grants the service account the right to impersonate specific users. You set it up in the Admin console, under security, then API controls, then manage domain-wide delegation. You enter the service account's client ID and a comma-delimited list of scopes.



Corn

How long does that take to propagate?



Herman

Google's own wording is that it usually takes a few minutes, but in some cases it might take up to twenty-four hours.



Corn

That's a fun window to debug in.



Herman

It's the classic. You change a permission, nothing happens, you assume you did it wrong, you change three other things, and then the first one kicks in and you have no idea which fix worked.



Corn

So if I'm Rita, why not just hand the app the super-admin's own credentials and skip the delegation ceremony?



Herman

Because delegation is a much narrower thing than it sounds. This is the misconception worth killing early. Domain-wide delegation does not grant the service account blanket access to all user data.



Corn

It sounds like it does.



Herman

The wording from Google is that you're not granting direct access to all user data, you're authorizing the service account to impersonate specific users when making API calls. So for every request, the application has to specify which user it's acting as, and it's bounded by two things at once. That user's own permissions, and the scopes you authorized.



Corn

So it can only do what Rita could do, and only within the scopes she signed off on.



Herman

Which is the right shape. If Rita can't see a folder, neither can the app acting as her.



Corn

And the alternative, just signing JWTs by hand?



Herman

Google actively pushes you away from it. The mechanics of server-to-server auth require you to create and cryptographically sign JSON Web Tokens, and the documentation says mistakes can cause serious errors that may have a severe impact on the security of your application.



Corn

That's a polite way of saying you will leak something.



Herman

Use the client libraries. That's the guidance.



Corn

So we have two auth models. Now the constraint that actually decides the Drive design.



Herman

Service accounts can't own files in Drive. That's straight from Google's folder guide. Service accounts can't own any files. They have to upload into shared drives, or use OAuth 2.0 to upload on behalf of a human user.



Corn

So the app can't just make a folder and be the owner of it.



Herman

It cannot. And that forks the whole design in two.



Corn

Walk me through the fork.



Herman

Path one, shared drives. A shared drive is owned by the organization rather than an individual, which is exactly what you want for client work, because the files don't belong to whoever happened to create them.



Corn

And the catch?



Herman

Shared drives need a paid Workspace edition. Business Standard, Business Plus, Enterprise, Education. A personal Gmail account cannot create one. So if Rita's on a paid Workspace tier, this is the clean path.



Corn

And if she isn't?



Herman

Then you go the other way. The service account impersonates a human user through domain-wide delegation, and the files get created as if Rita or her colleague made them.



Corn

Which works, but now every file in the system belongs to a person.



Herman

And people leave, or change roles, or get suspended while they're on holiday. A file owned by an individual is a file with a person-shaped single point of failure attached to it.



Corn

That's the kind of thing you only notice the day it matters.



Herman

There's a third wrinkle. Folders aren't special objects. A folder is just a file with a specific MIME type. Application slash vnd dot google-apps dot folder.



Corn

The folder is a file.



Herman

It's a file with no content and a type that tells Drive to draw it as a folder. You create it with a files.create call, you give it a name, and to nest it you set the parents property to the parent folder's ID.



Corn

One parent per file?



Herman

One parent. So it's a tree, not a graph. Which for Rita's client and project layout is fine, but it does mean you can't have a folder that lives in two places.



Corn

So the provisioning pattern, concretely. Rita adds a new client in the backend.



Herman

The backend makes a files.create call for the client folder, reads the returned folder ID, then makes calls for the subfolders underneath it, setting parents each time. And for every folder it creates, it captures the ID and writes that ID into the client row in Postgres.



Corn

So the database stores the folder ID, not the folder.



Herman

The identifier and the relationship. The bytes stay in Drive. Nothing gets duplicated. And if you ever need to find the folder again, you look up the ID you already have rather than searching Drive by name.



Corn

Which matters, because folder names aren't unique.



Herman

They're not, and two clients can both have a folder called site plans. The name is for humans. The ID is for the database.



Corn

What about limits? Two people, so probably not a concern, but.



Herman

Enormous on paper. Five hundred million items per user, five hundred thousand items per folder, a hundred levels of nested folders in My Drive. For a two-person consultancy that's effectively unlimited. But the hundred-level cap is a good reason not to let the tree grow organically, because somewhere around level forty someone has built a maze.



Corn

Let's talk money, because the pitch of this whole thing is that it stays cheap.



Herman

Cloud Run plus Cloud SQL Postgres is described over and over as the boring default for small and mid-sized GCP workloads. Cheap, scales to zero, integrates cleanly with IAM.



Corn

And the Postgres tier pricing?



Herman

A db-f1-micro instance, which is point two of a vCPU and point six of a gig of memory, list price is about eleven dollars and thirty-two cents a month. If you jump to a two-vCPU, seven and a half gig instance, you're at about a hundred and twenty-two dollars and forty-nine cents a month.



Corn

So the small one costs less than lunch.



Herman

There's a practitioner report of a Cloud Run plus Cloud SQL tenant app running at eighteen to twenty-two dollars a month. That's a realistic small-firm bill, not a theoretical one.



Corn

Cloud Run's cold start is what, a couple hundred milliseconds?



Herman

Around two hundred milliseconds, and it scales to zero, so you pay nothing when nobody's working. Cloud SQL doesn't scale to zero, it's a provisioned instance, but that's the trade you make for having Postgres.



Corn

And the connection limit is where people get bitten.



Herman

It's fixed by tier. A two-vCPU custom instance gives you about a hundred connections. So you pool connections in the app, and if you're going to have a lot of concurrent workers, you put PgBouncer in front of it.



Corn

So the honest summary is that the entire business runs for the price of a couple of coffees a month, and the hard parts aren't financial.



Herman

They're architectural. The auth model, the file ownership constraint, and the maintainability question we'll get to.



Corn

One more piece of context before we move into the working day, because it changes the picture a little. The free API layer is becoming metered.



Herman

Google started adjusting Workspace API quotas on the first of May this year, starting with Gmail, Calendar and Drive, and introducing a standardized tiering model tied to the Workspace MCP server launch.



Corn

And the mechanics?



Herman

New projects created on or after that date get the new quotas. Projects that used the APIs between November last year and April this year keep the old quotas for at least sixty days. And later this year, with ninety days' notice, quota increase requests will require Cloud billing enabled, and usage above the standard daily thresholds will generate charges.



Corn

So what does that mean for Rita's volumes?



Herman

Nothing today. The standard Drive threshold is four hundred million quota units a day. She's nowhere near it. Google's own expectation is that less than one percent of active developers will need to move beyond the standard tier.



Corn

But the direction of travel is the point.



Herman

The direction of travel is that the API layer has a meter on it now. Not because Rita's about to hit it, but because the argument for keeping your business data portable gets stronger when the alternative has a bill attached.



Corn

Okay. Rita's day. New client, new project, and everything that follows.



Herman

Client record first. Name, contact, site address, budget band, the relationship stuff. That goes into Postgres, because that's the relational heart of the business.



Corn

And the folder tree gets provisioned.



Herman

Through files.create calls, one per folder, parents set as it descends, IDs captured as it goes. Client folder, then a project folder under it, then the standard subfolders. Drawings, site survey, correspondence, invoices.



Corn

And those IDs go into the client and project rows.



Herman

Foreign keys pointing at Google objects. That's the pattern for the entire episode. The row knows the ID, the ID resolves to a Drive folder, the folder holds the bytes.



Corn

Now the presentation, because this is the piece Rita actually asked about.



Herman

Google publishes the exact recipe. It's the Slides mail merge. You build a template deck with placeholder tags. Something like double brace customer name, double brace project address.



Corn

So the tags are sitting there in the slide text.



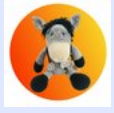
Herman

And then the workflow is three steps. Use the Drive API to copy the template. Then use the Slides API's presentations dot batch update call with replaceAllText requests to swap the tags for real values. Images use a separate request that auto-scales the image to the shape of the tag and preserves the aspect ratio.



Corn

That last part is the nice touch. The image quote fits the placeholder.



Herman

So the designers design a slide with a placeholder exactly the size and position they want, and the image lands in it correctly. No tuning pixel offsets in code.



Corn

And there's a warning attached?



Herman

There is, and it's in bold in Google's documentation. Once you've created a deck with tags in it, make a copy and use the Slides API to manipulate the copy. Do not use the Slides API to manipulate your primary template copy.



Corn

Because someone will, eventually.



Herman

Because if the API writes into the template, the tags get replaced, and the template is dead. The next client gets a deck with the previous client's name already baked in. Which is a spectacular way to send a proposal to the wrong person.



Corn

So the template is sacred and the copy is disposable.



Herman

And the reason this works for Rita specifically is in Google's own rationale. It's easy for designers to fine-tune a presentation's design using the Slides editor. Much easier than tuning parameters in your app to set the rendered slide design.



Corn

That's the argument. The designers keep editing in Slides, which they already know, and the app never tries to become a layout engine.



Herman

And after the merge, the copy is a completely normal deck. Rita's colleague opens it, moves things around, adds a plant schedule, and nothing about it is special.



Corn

What about when the merge fails partway? Half the deck has the new client's name and half still says the placeholder.



Herman

It doesn't fail that way. The batch update is all or nothing. If any request in the batch isn't valid, the entire request fails and nothing is applied.



Corn

So you either get a fully populated deck or you get an untouched copy.



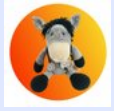
Herman

Right. Which is a much better failure mode than a half-edited document someone might accidentally send.



Corn

Docs and Sheets follow the same shape?



Herman

Same copy-then-replace pattern. Build a template doc or sheet, copy it with the Drive API, then substitute via each product's own batch update call. The tagging convention is the same.



Corn

That's how Rita's project briefs and her plant schedules get generated. Template, copy, fill, edit.



Herman

Now the messages and the events, and this is where the "what does the database store" question gets answered properly.



Corn

Which is barely anything, in the best way.



Herman

The database stores the link and the business meaning. The Gmail message ID, the thread ID, the Calendar event ID, the calendar ID. Plus metadata. Which project, which client, which milestone, what status it's in.



Corn

And the message bodies and attachments stay in Gmail.



Herman

They stay in Gmail. The event details, attendees, the video link, the notes, they stay in Calendar. Postgres holds the pointer and the meaning, Google holds the content.



Corn

So when Rita opens a project, the backend says here are the six threads and the four meetings attached to this landscape project, and it's not holding copies of any of them.



Herman

And that's the principle working exactly as intended. A single system of record doesn't mean copying everything into one place. It means one place decides what relates to what.



Corn

Now the sync question, because this is where the naÃ¯ve version falls over.



Herman

The naive version polls. Every five minutes, list the calendars, list the Drive folders, compare, update. And Google's guidance on this is unusually blunt.



Corn

Quote it.



Herman

An anti-pattern is to repeatedly poll every calendar of interest. This will very quickly use up all your quota.



Corn

So what do you do instead?



Herman

Push notifications. You register a webhook, a watch on files or on changes in Drive, and you get an empty POST callback with a header telling you what happened. Added, removed, updated, trashed, untrashed, change.



Corn

Empty as in no content?



Herman

Empty as in no metadata. The notification tells you something changed but not what. Your application then calls the Drive API using the resource ID or changes token from the notification to find out.



Corn

So it's a doorbell, not a summary.



Herman

It's a doorbell. And the channels expire. A files channel lasts at most one day, a changes channel at most a week, and there's no auto-renewal. You have to re-watch before it expires.



Corn

Which means there's a scheduled job whose entire job is to keep the doorbell alive.



Herman

A small one, but it has to exist. And it's exactly the piece that a scattered approach with a script here and a script there tends to miss, because nothing about it is visible until the notifications stop.



Corn

And the durable reconciliation mechanism?



Herman

The changes API. You call get start page token to get a starting point, save that token, and then list changes with it to enumerate everything that's happened since your last sync. That's how you catch anything the doorbell missed.



Corn

Belt and braces. The push notification tells you to look, the changes token tells you what you missed while you weren't looking.



Herman

Which matters because a webhook can fail. The callback URL goes down for an hour, the notifications queue up or get dropped, and you have no idea what you missed. The changes token is the answer to that.



Corn

Now the quota trap, because this one is nasty.



Herman

It is. With domain-wide delegation, by default the service account is charged against the per-minute, per-user, per-project quotas, and not the user you're impersonating.



Corn

Say that again, because it's counterintuitive.



Herman

The service account wears the quota, not the person. So if your app is acting on behalf of Rita and her colleague and every client they email, all of that traffic is being counted against one identity.



Corn

So it doesn't matter how many users you're serving, you've got one bucket.



Herman

One bucket. And a single service account can get rate-limited even while it's acting for many users. The fix is to pass the quota user parameter, either as a URL parameter or a header, on each request, which attributes the quota to the real user.



Corn

Is that documented, or is it tribal knowledge?



Herman

It's in Google's Calendar quota documentation. It's documented. It's just buried, and if you're writing a five-line Apps Script you'll never encounter it.



Corn

Whereas if you're building the integration layer once, you write the interceptor once and every call gets it.



Herman

That's the argument for the layer, honestly. Not that the layer is clever. That the layer is where the ugly details live, once.



Corn

What about handling the rate limit errors when they do happen?



Herman

Truncated exponential backoff. Wait two to the power of the attempt count plus a random number of milliseconds, capped at a maximum, which is typically thirty-two to sixty-four seconds. And the errors surface as a 403 with user rate limit exceeded, or a 429 with rate limit exceeded.



Corn

And the numbers, just so people know how much headroom there is.



Herman

Drive is a million quota units a minute per project and three hundred and twenty-five thousand a minute per user, with a daily billing threshold of four hundred million units. Per-method, a file get costs five units, a file list costs a hundred, a download costs two hundred, an update costs fifty.



Corn

So a list? A hundred units? Just to list.



Herman

Listing is expensive relative to getting. Which is another argument against the polling approach. You're spending a hundred units every time you ask what's in a folder.



Corn

And Gmail?



Herman

A million two hundred thousand units a minute per project, six thousand a minute per user, daily billing threshold of eighty million. A messages list is five to ten units, a get is twenty, a send is a hundred, a watch is a hundred.



Corn

Calendar is request-based rather than unit-based, right?



Herman

Ten thousand requests a minute per project, six hundred a minute per user for projects created after May this year, and a daily billing threshold of a million requests.



Corn

For Rita, all of this is headroom she'll never touch.



Herman

She'll never touch it. The point isn't the ceiling. The point is that all of this is the kind of thing that has to be handled in one place, and the maintainability argument is entirely about putting it in one place.



Corn

Make the case. Why a reusable integration layer instead of an Apps Script per task?



Herman

Because look at what we've listed. The service-account quota trap. The channel expiry re-watch job. The changes token reconciliation. The copy-then-replace discipline on templates. The backoff handling. Each of those is a thing that has to be right, and each of them is invisible until it's wrong.



Corn

In the scattered model, each script gets its own answer.



Herman

Each script gets its own answer, or more likely its own version of the same mistake. And then six months later you have eleven scripts, four of which are doing something subtly different, and nobody can tell which ones still run.



Corn

The way it goes wrong isn't the script that breaks. It's the script that quietly does the wrong thing and nobody notices.



Herman

The script that stops re-watching the Drive channel because the cron job died, and the sync has been stale for three weeks, and everyone assumes it's working because nothing errored.



Herman

We did cover the friction between Workspace apps and Google Cloud tools for a solo consultant, a while back. The tension hasn't changed. The tools want to be used the way their designers imagined, and the business wants them used its own way.



Corn

The fix is a translation layer, not a fight.



Herman

The translation layer. Which is the backend. You build the operations once. Provision a client. Create a project folder tree. Generate a proposal deck from a template. Link a thread to a milestone. Attach an event to a project. Reconcile Drive changes into Postgres.



Corn

Each of those is a function the app calls, not a script sitting in a spreadsheet.



Herman

The app doesn't know about JWT signing. It doesn't know about backoff. It calls the operation, the operation deals with Google, and the details live in one place where they can be fixed once.



Corn

Rita keeps her Workspace investment exactly as it is. The designers still work in Slides. The colleague still works in Gmail. And there's a structured layer underneath that actually knows what relates to what.



Herman

Which is what she asked for. Not a replacement. A spine.



Corn

One thing worth flagging before we get to Hilbert, and it's the honest open question. The May quota change means the free API layer is becoming metered, and the design fork on shared drives depends on which Workspace edition Rita's actually on.



Herman

If she's on Business Standard or above, the shared drive path is clean, the files belong to the organization, and there's no person-shaped single point of failure. If she's not, the app impersonates a human user and every file has an owner with a name.



Corn

Which one she picks changes how much of this she can hand to a system versus how much stays attached to a person.



Herman

It's a real fork. It's not a detail.



Corn

Okay. Let me make one prediction about how this is going to go, because I've watched you all episode.



Herman

Go on.



Corn

You're going to say the folder structure is the hard part.



Herman

The folder structure is the hard part.



Corn

You were never going to make it through without saying that.



Hilbert

The folder structure is fine. The problem is what people name them.



Hilbert

I've seen it. I knew a man in the licensing office, Eitan, who ran a shared drive for a department of twenty-three people. Every project folder was named the same way. Two letters, a hyphen, and a number.



Herman

Which two letters?



Hilbert

The initials of the person who was going to use it.



Corn

The folder is named after a person.



Hilbert

Every folder. And when the person left, the folder kept their initials, because nobody wanted to be the one who renamed it and broke everyone's shortcuts. By the time I got there, fourteen of the twenty-three folders were named for people who had not worked there in years.



Herman

The naming scheme was a personnel directory.



Hilbert

The two folders at the top, the ones everyone actually needed, were named after a dog. A retired dog. It had worked in the department, apparently.



Corn

The dog worked in the licensing office.



Hilbert

It came in. It was on the org chart. Nobody was going to be the one to take it off.



Herman

How long did this go on?



Hilbert

Eleven years. I was there for the last four of them. And underneath all of it, at the bottom of every project folder, was the one folder that mattered. It was called, in full, DO NOT MOVE.



Corn

What was in it?



Hilbert

The current version of everything.



Herman

Everything?



Hilbert

Every drawing that was still live. There were six hundred and forty of them. Because the way the system had grown, nobody trusted the version in the project folder, so the working copy went into DO NOT MOVE, and the project folder held the old one.



Corn

The folder called do not move was the only folder anyone moved things into.



Hilbert

Constantly.



Herman

Did you fix it?



Hilbert

I renamed it. First week. It's called CURRENT now.



Corn

And?



Hilbert

For about two years afterwards, people would open the drive, look for DO NOT MOVE, not find it, and email me to say the drive was broken.



Corn

What happened to the dog's folder?



Hilbert

Still there. That one I left alone.



Herman

The one thing you didn't touch was the folder named after a dog.



Hilbert

The dog had a plaque. You don't rename a folder that has a plaque.



Herman

The principle we've been describing all episode is that the database maps the structure and Drive just holds it. What Hilbert's describing is the version where the structure was never mapped anywhere, so the folder names became the only record of anything.



Corn

The names are the worst possible place to keep a record, because names are what humans edit on a Tuesday afternoon.



Hilbert

The names are what you change when you're annoyed.



Corn

That's the piece I'll take from it. If the structure only exists in names, then every rename is a data migration nobody ran.



Hilbert

That's not how I'd say it, but yes.



Herman

The folder IDs don't care what anything is called. Rename the client folder tomorrow and the row in Postgres still points at the same folder.



Corn

Which is why the ID goes in the row and the name doesn't.



Hilbert

Now you've got it.



Corn

The open question for Rita, and this is where the whole series is heading, is what happens when the free API layer stops being free and she's got a business spine that depends on it.



Herman

The thresholds are enormous, so it isn't a next-month problem. It's a next-few-years problem. And the answer is the same one that's been running under this whole episode. Keep the relationships in Postgres, let the external services be authoritative for what they host, and make sure you could move a piece of it without moving all of it.



Corn

Which is what Part Three is going to stress-test, because WhatsApp and meeting transcription and external CRMs and AI agents are all services that would very much like to be the only place your data lives.



Herman

The principle holds. Postgres owns what relates to what. Everything else owns what it made.



Corn

Thanks to Hilbert Flumingtop for producing, and for the eleven-year drive tour.



Herman

For more along these lines, there's episode thirty-five, The Privacy Gap; episode three forty-seven, Bridging the Workspace-GCP Divide; and episode seventeen fifty, Herman's Music Showcase. This has been My Weird Prompts.



Corn

If you want to send us a prompt, send us your own prompt on Telegram at t dot me slash MWP listener bot. And if you've been enjoying the show, a review helps people find it.



Herman

We'll be back soon.



Corn

See you tomorrow.