Episode Book

EPISODE #6618

How Label Studio's XML Configs and Storage Work

PUBLISHED	15 November 2026 at 20:21 UTC
DURATION	26:48
LISTEN	myweirdprompts.com/episode/label-studio-xml-self-hosting-11211/
HOSTS	Corn, Herman, Dorothy, Hilbert
SCRIPT MODEL	z-ai/glm-5.3
VOICES	omnivoice
PIPELINE	V5.33
SOURCES	14 cited by the research (listed at the end)
BOOK GENERATED	15 November 2026 at 23:00 UTC

SYNOPSIS

Label Studio powers everything from bounding boxes to audio transcription with one design decision: an XML file that wires data types to annotation controls. This walkthrough covers how HumanSignal's open-source tool separates free individual labeling from paid team coordination, and what self-hosting to the Hugging Face Hub actually looks like. Expect the real architecture behind cloud storage syncing, pre-signed URLs versus proxy mode, and why pushing to the Hub still requires exporting first.

TRANSCRIPT



Corn

Every few months someone declares data labeling solved. The models will label themselves, the vendors will handle it, nobody will ever squint at a bounding box again. And every few months the same quiet thing happens: a human being opens a web form and draws boxes on images until their wrist hurts, because that's still the only way.



Herman

And the tool that human is most likely to be staring at is Label Studio. Which leads us to Daniel's prompt, because he's been staring at it himself — he self-hosts it, and he wants the proper runthrough, not the brochure version. Two halves to what he's asking. First, why has this one open-source tool become the default for annotating text, audio, images, video, LLM outputs — what's the actual mechanism behind the flexibility. And second, the practical path: you're self-hosting, you've got a dataset destined for the Hugging Face Hub, and you want to know where the media actually lives, what the database holds, how storage syncing works when the dataset is big, and how the annotations get out and up onto the Hub.



Corn

He said the words I always want to hear: "the practical side, not the marketing."



Herman

Which is convenient, because the marketing is the least interesting part of this thing. The interesting part is that the whole product is one design decision, and everything else falls out of it.



Corn

Start there, then.



Herman

Label Studio came out of a company called HumanSignal — the repo was originally under Heartex, and the first push was June of twenty nineteen. Apache-2.0 license, about twenty-eight thousand seven hundred GitHub stars, written mostly in TypeScript with a Python and Django backend. And it's alive: stable release at the end of September, version



Herman (continued)

1.23.2, and the last commit to the repo was yesterday. This is not an abandonware situation.



Corn

Rare enough in open source annotation tools that it's worth saying out loud.



Herman

It is! The graveyard in that category is crowded. Their own README pitches it plainly: label audio, text, images, videos, time series with a simple UI and export to various model formats — either preparing raw data or improving training data you already have. And for a self-hoster the install paths are the usual menu: pip install if you've got Python 3.10 or newer, a single docker run that drops the SQLite database and your uploads into a local folder, Docker Compose with Nginx and PostgreSQL if you want the grown-up version, a Compose variant that adds MinIO so you can test S3-style buckets locally, and one-click buttons for Heroku, Azure, Cloud Run.



Corn

The single docker run is the one that catches people, because everything lands in one folder — database, uploads, all of it.



Herman

Which is a feature until the day you delete the container without mounting a volume, and then it's a very short conversation with yourself. Two questions for today, then. Why is this thing the default, and what does the self-hosted-to-Hub path actually look like end to end. I'd take the first one, because the answer is elegant and I've been waiting to talk about it.



Corn

Go on. You've been vibrating since the prompt came in.



Herman

The labeling interface is not code. That's the whole thing. Every project's interface is defined by an XML file — the labeling config — built from three families of tags. Object tags declare the data type and where it comes from — an image tag with a value pointing at a task field. Control tags declare what the annotator produces — choices, labels, rectangle labels, brush labels, keypoint labels, polygon labels, a text area. And visual tags just lay the screen out. A control tag binds to an object tag through a toName attribute, and that binding is the entire wiring.



Corn

So named entity recognition and drawing polygons on a CT scan are the same product with different XML.



Herman

Same engine, different config. Text plus labels, you've got NER. Image plus rectangle labels, object detection. Brush labels, segmentation. Audio plus a text area, transcription. Video, time series, multimodal — each one is a different config, not a different product. That's why it covers everything Daniel listed. And here's the part that matters for the Community-versus-paid question: the tag library and the templates are available in all three editions, including free Community. The flexibility is not paywalled.



Corn

That's the detail I'd want flagged in neon, because everyone assumes the free tier is a stripped demo.



Herman

What's actually Enterprise-only on the interface side is the programmable React-embeddable stuff and JavaScript plugins — building the labeling UI into your own application. Which is a real product, but it's not labeling.



Corn

It's for companies whose annotators must never learn what a browser address bar is.



Herman

The tooling around configs has matured too, though. HumanSignal ships an agent skill now — you describe the task in plain English, it drafts the XML, validates it against your running instance, and after you explicitly approve, pushes it as a new project or an update. There's a community repo of shareable configs. And the SDK has a shortcut so you don't hand-write the name and toName wiring — you give it a couple of fields and choices and it builds the config for you.



Corn

An agent that writes XML for you, so you can label data to train agents. The circle closes.



Herman

One caveat worth carving somewhere visible: if you change a project's config while annotation work is in progress, you may have to delete the existing annotations. So the config is a real upfront design decision, not something you live-edit your way into. Get it wrong with ten thousand tasks labeled and you'll meet the config editor under very different emotional circumstances.



Corn

So the config is the product — which raises the obvious question of what you actually pay for.



Herman

This is my favorite part of the whole comparison table, because it's cleaner than most. Community — free, self-hosted — includes the Data Manager, all data formats, import from database, cloud, or local, pre-annotated predictions, export to JSON, COCO, Pascal VOC, sync between projects and external storage, templates, the tag library, the API and SDK, webhooks, and custom ML backends. That is a complete tool.



Corn

What's left to sell?



Herman

Managing people. Role-based access, workspaces, task assignment, bulk labeling, active learning loops, dashboards for projects and annotators, activity logs, agreement metrics, single sign-on, SOC2, an uptime SLA, white labeling. None of that helps you label a single image. All of it helps you run a floor of thirty annotators and a reviewer who checks their overlap.



Corn

A solo self-hoster reads that list and feels nothing.



Herman

Daniel reads that list and feels nothing. He's one guy with a server. The wall appears when there's a team and a review process — the moment the work is about coordinating humans, not tagging pixels. Now, the paywall isn't perfectly clean, and I want to be honest about where it bites. The LLM features are the sharpest edge. The Prompts features — automated pre-labeling, LLM fine-tuning and evaluation, benchmarks — are absent in Community, a limited trial in the cloud Starter tier, full in Enterprise. The Chat feature, for generating and labeling conversation data, same ladder. So if you're buying Label Studio specifically for its LLM story, Community doesn't have one.



Corn

And the storage side?



Herman

Advanced cloud storage auth is Enterprise-only — S3 with IAM roles, GCS Workload Identity Federation, Azure with Service Principal. But plain S3, GCS, Azure Blob, Redis and local storage all work in Community, and that's honestly enough for most self-hosters. The fancy auth matters when your labeling platform lives inside a corporate cloud with an identity perimeter. Daniel's instance is presumably not doing GCS service-account impersonation for GKE.



Corn

His instance is doing well to be backed up.



Herman

I'll have you know I set up backups for my own projects after breaking them. There's a pattern with me, I don't deny it. Where was I — right. The one-line version of the whole edition question: labeling is free, coordination is expensive.



Corn

All of that is the why. Let's get to the how — you've got a dataset, you're self-hosting, and you want it on the Hub.



Herman

And the first thing to say is the thing Daniel will already have discovered if he looked for a button: there is no push to Hugging Face button. The Hub is not in the storage-connection table at all — that table lists S3, GCS, Azure Blob, Databricks, Redis and local storage. Every push-to-Hub path that exists is export, convert, push, whether that's an SDK script, git with LFS, or a third-party CLI. So the round trip is real, it's just not native.



Corn

Which is almost flattering. Everything else integrates with the Hub; the labeling tool is the one holding out.



Herman

Give it a second before judging, because the newest approach does something strange. But first — the question Daniel is really asking underneath all of this: where do the media files actually live. And the docs answer it in one sentence that deserves to be quoted: Label Studio does not import media files from your source storage. Media stays in your bucket and is loaded through pre-signed URLs or the storage proxy when a user opens a task.



Corn

That's the single most important architectural fact for anyone with a big dataset.



Herman

It completely changes your mental model. You're not uploading a hundred thousand images into Label Studio. You point it at a bucket, and it references. What lands in the database depends on the import method you pick when you connect the storage. In Files mode, Label Studio lists the bucket objects and constructs one task per file, and each task stores only a URI reference — the backend needs LIST permission and downloads nothing. In Tasks mode, it reads your JSON or JSONL task files and saves the task data — text, metadata — into the database, and the backend needs GET permission. Tasks mode is for complex tasks with multiple media sources per task, and one rule bites people: you can put multiple tasks in a JSON file, but you must not mix task formats in one file.



Corn

The permission split is a nice tell, actually. LIST-only means the labeling platform can't even read your data.



Herman

Just enumerate it. It's a defensive design. Now the surprise, and I know it surprised me: sync is manual. Label Studio does not automatically sync data from source storage. You upload new objects to the bucket, and nothing happens — you trigger a sync from the UI or the API, and then the new tasks appear. No restart needed, but no polling either. Source storage is really two halves: the sync, which scans storage and imports tasks, and URI resolving, which turns an S3-style path into an HTTPS URL a browser can actually load.



Corn

Herman, hold on. Someone uploads to the bucket and the tool just... waits politely to be told.



Herman

Forever. It will never notice on its own. It sounds like an omission until you've run an auto-syncing pipeline, and then it sounds like mercy. Next fork, and this one has real consequences: pre-signed URLs or the proxy. Pre-signed is the default — the browser gets an HTTP 303 redirect to a time-limited URL straight from S3 or GCS or Azure. Media stays isolated from the Label Studio network entirely, which is clean, but you need CORS configured, presign permissions, and the browser needs HEAD as well as GET, because that's how it learns the file size for audio and video playback.



Corn

And the other fork?



Herman

Proxy storage — you toggle pre-signed URLs off, and the Label Studio backend fetches the objects server-side and streams them to the browser. Everything stays inside your network boundary, one entry point, no CORS, no presign permissions — you need read object and list bucket and that's it. And the access control follows the user: revoke someone's task access and the cached file is blocked for them too. The cost is that every byte of every image flows through your backend.



Corn

So it's the difference between handing out time-limited keys and running everything past the doorman.



Herman

That's actually not a bad compression of it. Target storage is the reverse direction — you attach target connections to sync annotations out of Label Studio into external storage, and both source and target are project-specific, and you can hang multiple buckets, containers, databases or directories off one project. When it's a cloud bucket, you get one file per labeled task, named task underscore id dot json.



Corn

Well, that's the storage side of things. Now let's hear from your mother, who has something from the history of toast.



Herman

Oh, it's Dorothy's History of Toast! Yes, please.

D

Dorothy

It's Dorothy, dear, with Dorothy's History of Toast.

D

Dorothy

Mrs Beeton's Book of Household Management, from eighteen sixty-one, has a recipe for a toast sandwich. One slice of toast between two slices of bread. Recommended for invalids.

D

Dorothy

You'd think toast would be a comfort, and then it turns out it's mostly for people who are already having a hard time of it.

D

Dorothy

Anyway. I've got a bit of ironing waiting, sweetheart, so I'll let you get on. Bye bye now.



Corn

A toast sandwich. Mum's out here documenting the nineteenth century's saddest lunch.



Herman

I'd eat one. That's not the point, but I would. Back to the database, though — because there's a second half to Daniel's question about what the database holds versus the bucket. The database holds the tasks, the annotations, the accounts. And the default is SQLite, which sounds alarming and mostly isn't: the docs say it works well for projects with tens of thousands of tasks, as long as you don't need complex Data Manager filters or heavy multi-user pipelines.



Corn

And the threshold where it stops working?



Herman

They're explicit about it. Above a hundred thousand tasks with five or more concurrent users, use PostgreSQL. And if you don't, the symptom is wonderfully concrete — task loads and annotation saves can take more than ten seconds. Which, for an annotator, is death by a thousand loading spinners. One more version note: as of 1.23.1 you need PostgreSQL fourteen or newer, because the Django upgrade dropped thirteen.



Corn

Ten seconds to save a checkbox. That's not a tool anymore, that's a personality test.



Herman

So the media never moves. What does move is the annotations — and that's where the self-hosting cliff shows up. Community export is synchronous. Generated as part of the request, no background export workers by default. And a synchronous export of a large project will blow straight past your reverse proxy's timeout — commonly around ninety seconds — and come back as a 502 or 504. That's the actual cliff in self-hosting. Not storage, not the database. The export.



Corn

The free tier labels a million images and then can't hand them to you.



Herman

It hands them to you if you're patient and you don't ask through an nginx timeout. There are workarounds, all real ones. The SDK can export snapshots. There's a console command — label-studio export, project id, format, output path — which runs in a terminal where ninety seconds is nothing. And there's the Snapshot API: you POST to create a snapshot, get an ID back, poll its status, then GET to download. Enterprise gets background workers and asynchronous snapshots, and that's a paid feature doing paid work.



Corn

Before the convert step, what's actually inside an export? Because this is where the gotchas live.



Herman

Annotations are raw JSON. Each one contains regions — the span, the box, the segment — and results, the labels on that region. The region IDs are strings, letters, digits, underscore, hyphen, and each result's ID matches its region's ID, which is how you trace a model's prediction through to the human correction of it. That's a quietly beautiful detail, honestly — the correction carries the lineage of the thing it corrected.



Corn

And the gotcha?



Herman

Image annotations are stored as percentages of overall image size. Not pixels. Every conversion to COCO, YOLO, or an HF image format has to multiply by the actual dimensions, and every few months somebody on a forum discovers their boxes are all in the top-left corner because they skipped that step.



Corn

Classic. What else trips people at the export door?



Herman

The Easy Export API excludes unannotated tasks by default — you have to ask for all tasks explicitly. And UI export ignores your tab filters: it always includes annotated tasks, and cancelled annotated tasks come along too. So the export is honest, but it's honest about things you filtered out on purpose.



Corn

So the media never moved, the annotations come out as JSON, and the Hub is on the other side of a conversion script. What are the actual routes?



Herman

Three, roughly, in chronological order. The oldest canonical recipe is Daniel van Strien's, from twenty twenty-two — and yes, different Daniel, which the archive finds funny every time. You export annotations from Label Studio, load them into a pandas DataFrame, and then you use the Hub as what it is underneath: a git repository. Track the JPGs with git LFS, add, commit, push. Images and annotation CSVs live together with full version history.



Corn

Git LFS. The file format equivalent of a van with a trailer.



Herman

It works, and it's the version I'd trust with my own data, because I understand every part of it. Second route: HumanSignal's own tutorial does the full round trip with the SDK. You install the SDK plus datasets, transformers, torch, the Hub library, and accelerate. You connect with a client using a personal access token from Account and Settings. You build an NER config, load a hundred rows of WikiANN, join the tokens into a text string, and wrap each row as a task with the text in data and the source and index in metadata so you can find your way home again. Import via SDK, label, export via SDK, convert to a datasets Dataset.



Corn

And the tutorial's numbers?



Herman

Sixty to eighty percent faster labeling, and a ten-thousand-document project dropping from five hundred thirty hours to a hundred thirty. Vendor claims, their own math, no independent measurement anywhere. I'd file those under marketing and move on.



Corn

Third route.



Herman

The newest and the strangest. Van Strien again — a skill repo that deploys Label Studio onto a Hugging Face Space, with an HF Storage Bucket mounted at slash data so the SQLite database and uploads survive Space restarts. Three CLI commands: create the bucket, duplicate the Label Studio Space with the bucket mounted, factory reboot. Then you push tasks in from a Hub dataset, post model predictions as prelabels — with the pixel-to-percentage conversion handled — and there's a script that exports annotations back out to a Hub dataset. Auth is a refresh token, and the scripts auto-refresh the JWT, which only lasts five minutes. It's Apache-2.0 and it says proof of concept on the tin, in those words.



Corn

So self-hosted and on-the-Hub collapse into one thing. Your labeling tool is a Space, your data is a bucket, your output is a dataset.



Herman

That's the pitch — annotation infrastructure on the same platform as your datasets, your models, your batch inference, no SaaS in the loop. It's the convergence answer to a question nobody standardized. And there's a third-party CLI, labelr from Open Food Facts, that exports straight to a Hub repo — one command, from Label Studio to Hugging Face — with a limitation stated plainly in its README: it can't pull your label names out of Label Studio, you have to pass them yourself.



Corn

A push-to-Hub tool that doesn't know your labels. Half a bridge, honestly.



Herman

It's a volunteer CLI for a food database, I cut it slack. But the shape of the whole landscape is three people solving the same problem three ways, and none of them is a connector in the storage table.



Hilbert

It's presigned. One word. You've both been saying "pre-signed" with a hyphen all afternoon and I filled in a suppliers' form in nineteen ninety-four where they crossed that out in red, and I've written it correctly since.



Herman

Noted for the transcript.



Hilbert

The rest of it you had right. My neighbour Reg has a moth survey — forty thousand photographs, wing patterns, wants them classified for a model. I did the labeling, on my own install, community edition. The export was the problem. Anything over about ninety seconds died at the proxy, same as you said. Reg's dataset, my evenings.



Herman

So how did you get the annotations out? Snapshot API?



Hilbert

I did it by hand, in the end. Batches of two hundred tasks, export each one, and I wrote the batch numbers on the back of a seed catalogue in pencil so I'd know where I'd got to. The interface isn't built for claws, so a batch took me the better part of an hour. Took nine nights.



Corn

Nine nights, two hundred tasks a time. You were the background export worker.



Hilbert

I was cheaper than the enterprise licence. The converted dataset's on a drive at Reg's house — he labelled it Moths, Final, which it wasn't, because he sent back three batches. Reg is thorough, I'll say that for him. He sat with every rejected batch himself.



Herman

That's the part the comparison table can't price.



Hilbert

No. He did the work properly and didn't complain about once. Anyway, your ninety-second figure is right, so somebody at that company knows what happens at the proxy.



Corn

The seed catalogue still has the numbers on it, I assume.



Hilbert

Naturally. That's what it was for.



Corn

Reg's handwriting on the drive and Hilbert's pencil on the catalogue. That's a dataset with provenance, technically.



Herman

Provenance nobody asked for but everybody got. So the export cliff is real even at the kitchen-table scale — the Snapshot API would have saved him eight nights, but the batches worked, and Reg got his moths.



Corn

He got most of his moths, then rejected some on principle.



Herman

The fine-tuning episode we did ages ago — the whole question there was what the model actually learns from. This is the answer upstream of it: it learns from Reg, checking three returned batches at his kitchen table. Anyway. Cutting-room floor — two things I couldn't fit. The export formats list includes ASR manifest, the NVIDIA NeMo format, which is a nice touch for a general tool — it'll hand you audio transcriptions shaped for a specific speech stack. And the Docker container runs as a non-root user, UID one thousand and one, so your mounted volume needs matching permissions or the thing sulks at startup.



Corn

The sulking is how you learn the UID exists.



Herman

It's a very educational sulking. Where I'd leave the open question, though: whether a native Hugging Face Hub connector ever lands in that storage table. The Spaces skill is the closest thing to a hint, and it's labeled proof of concept by its own author — so the direction is visible, the commitment isn't.



Corn

And one flag before we go, for anyone running an old pip install: the September release notes mention that older versions wrote the secret key to a dot-env file in the data directory, world-readable. Rotating it logs everybody out and invalidates access tokens — which is exactly what it's for.



Herman

Worth knowing before you rotate, not after. And the deeper point stands, I think: the Community line is coordination, not capability — the real cliff in self-hosting is a synchronous export, not the storage or the database. If you're one person with a bucket, you have the whole tool.



Corn

Thanks as always to Hilbert Flumingtop at the desk, and to the excellent Dorothy.



Herman

If you enjoyed this one, try episode six, How To Fine Tune Whisper. This has been My Weird Prompts. If the show earns a review from you, we'd take it — it's how people like us get found.



Corn

Find us at my weird prompts dot com. We'll be back soon.

SOURCES

What the research for this episode read before the script was written. Primary sources first.

1. **GitHub repo, README, releases 1.23.1/1.23.2/nightly; retrieved 2026-11-15; latest stable v1.23.2 (2026-09-29). [primary]**
   https://github.com/HumanSignal/label-studio

2. **Community vs Starter Cloud vs Enterprise feature table; retrieved 2026-11-15. [primary]**
   https://labelstud.io/guide/label_studio_compare.html

3. **Cloud storage setup, source/target storage, import methods, presigned vs proxy; retrieved 2026-11-15 (truncated before target-storage detail). [primary]**
   https://labelstud.io/guide/storage.html

4. **raw markdown of the same doc, `develop` branch; retrieved 2026-11-15. [primary]**
   https://raw.githubusercontent.com/HumanSignal/label-studio/develop/docs/source/guide/storage.md

5. **database setup, SQLite vs PostgreSQL sizing, MinIO, persistence; retrieved 2026-11-15. [primary]**
   https://raw.githubusercontent.com/HumanSignal/label-studio/develop/docs/source/guide/storedata.md

6. **export formats, synchronous Community export, snapshot API, console command, `task_id.json`; retrieved 2026-11-15. [primary]**
   https://labelstud.io/guide/export.html

7. **HumanSignal HFLS SDK tutorial (WikiANN NER); retrieved 2026-11-15. [primary]**
   https://labelstud.io/tutorials/how_to_connect_hugging_face_with_label_studio_sdk

8. **labeling config: object/control/visual tags, `toName`; retrieved 2026-11-15.**
   https://labelstud.io/guide/setup

9. **`create-xml-labeling-config-skill`; retrieved 2026-11-15.**
   https://docs.humansignal.com/skills/xml-config

10. **`LabelInterface.create` / `create_simple`; retrieved 2026-11-15.**
    https://github.com/HumanSignal/label-studio-sdk/blob/master/src/label_studio_sdk/label_interface/interface.py

11. **van Strien, Label Studio x Hugging Face datasets hub, 2022-09-07; retrieved 2026-11-15.**
    https://danielvanstrien.xyz/posts/post-with-code/label-studio-hub/2022-09-07-label-studio-annotations-hub.html

12. **deploy LS on HF Spaces + export annotations to Hub; proof of concept, Apache 2.0; retrieved 2026-11-15.**
    https://github.com/davanstrien/label-studio-on-spaces-skill

13. **third-party CLI with `--to huggingface` export; retrieved 2026-11-15.**
    https://github.com/openfoodfacts/labelr

14. **community config collection; retrieved 2026-11-15.**
https://github.com/humanSignal/awesome-label-studio-config