LF AI & DATA

Powering batch analytics and real-time applications with LF AI & Data projects Kedro and Feast

How Every Cure built a reusable feature platform for biomedical machine learning

kedro

FEAST

At a Glance

Team
Every Cure, biomedical research non-profit that applies machine learning to systematically identify drug repurposing candidates

Tool Integrated
Kedro (LF AI & Data)
Feast (LF AI & Data)

Use Case
A reusable feature platform serving both batch analytical pipelines and real-time applications

Intended Users
Internal application reviewers, analytical pipelines, and agentic workflows

Store Architecture
Offline store (BigQuery) for training and analytics; online store (Postgres) for low-latency serving

Feature Lifecycle
Register, write, materialize, read

Compute Engine
Local by default; Spark available for larger, distributed workloads

Orchestration
Feast apply automated on pull-request merge; feast materialize-incremental run on a daily CRON job

Author
Laurens Vijnck, Principal Engineer

Executive Summary

Every Cure applies machine learning to identify drug repurposing candidates: drugs that may treat diseases they weren't originally developed for. The organization builds and maintains large-scale data pipelines to engineer and serve the features that power its candidate review models.

Its pipeline integrates dozens of biomedical datasets into thousands of engineered features that help prioritize drug–disease pairs for medical review. Kedro powers these data pipelines, turning disparate biomedical sources into a unified feature layer. Kedro is an open source Python framework hosted by the LF AI & Data Foundation that brings reproducibility, modularity, and engineering discipline to modern AI and agentic workflows, helping teams transform ideas into clear, manageable pipelines for scalable AI solutions.

As the platform grew, the team found it hard to reconcile the batch-oriented output of its Kedro pipelines with the real-time needs of its application and agentic workflows. Adopting Feast, another LF AI & Data project and an open source feature store, closed that gap: Kedro and Feast together now give Every Cure a single feature registry that supports both batch pipelines and real-time applications, speeding up review of drug repurposing candidates and giving consistent feature access across analytical, application, and agentic workflows.

The Challenge: Two Systems, Neither One Enough

Every Cure needed a structured, standardized way to build data pipelines for large-scale workflows that transform disparate biomedical sources into a unified feature layer. Kedro became the framework of choice: its emphasis on reproducibility, modularity, and software engineering discipline matched the team's commitment to building pipelines that are auditable, maintainable, and scalable from the start.

Having adopted Kedro, the team then faced a new question: how to extend its pipeline model to serve real-time feature needs alongside batch ones.

Before adopting Feast, Every Cure ran two separate systems for feature access, one built for batch workloads, one for transactional access, and neither covered both use cases.

For batch workloads, a datasets repository versioned feature files in git: lightweight, auditable, and Kedro-friendly, but file-based only, with no entity-level view and no transactional access. For interactive access, an internal "evidence store" loaded a handful of features into Postgres so reviewers could filter and sort. It handled real-time queries but had no analytical capability, needed manual intervention to load data, and only understood a fixed set of entities.

The result was two half-answers to the same question: there was no single place to see every feature available for a given drug-disease pair, and every new entity or feature type meant making plumbing changes in two systems instead of one.

The Solution: Feast Behind Kedro

Every Cure adopted Kedro as the standardized framework for building its data pipelines at scale. Kedro provided the engineering backbone that made the rest of the work possible: a consistent way to structure, version, and run the pipelines that produce features across the organization.

With that foundation in place, the team could integrate a dedicated feature store — Feast, an open source tool for storing, versioning, and serving machine learning features — directly into their existing Kedro pipelines.

Every Cure chose Feast for its open source nature, its independence from any single cloud platform, and its comparatively low operational complexity. Features are defined once in a central location, allowing any Kedro pipeline to consume them, an approach that becomes especially valuable as multiple teams and projects begin producing and using the same features.

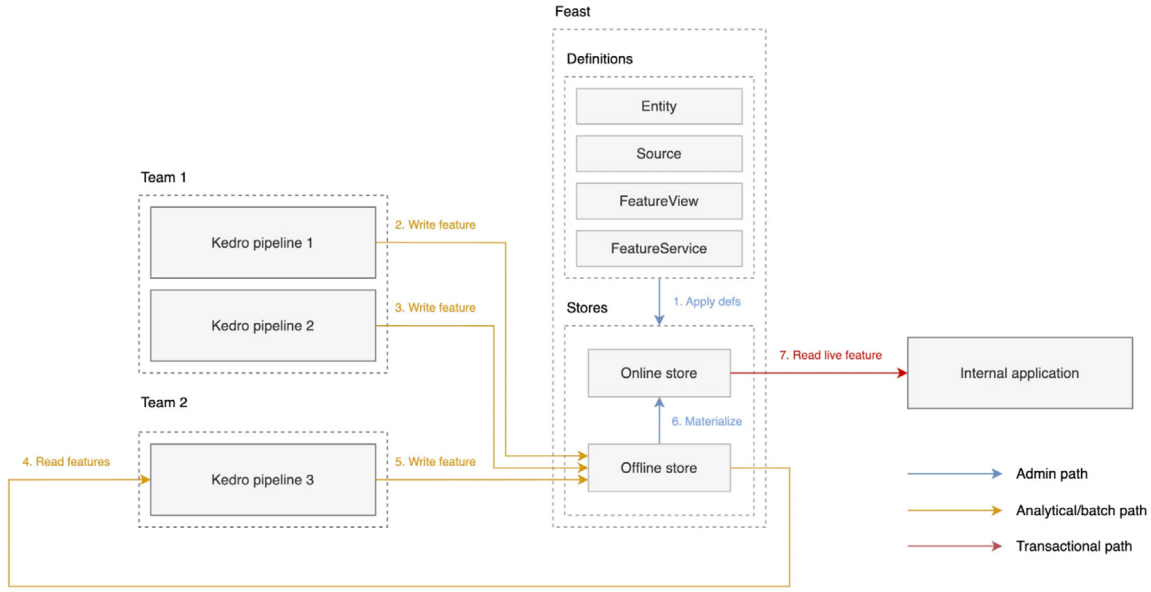
Feast's registry splits into two stores: an offline store for training and analytics, and an online store for low-latency serving. Feast itself runs no server, and it delegates to BigQuery for the offline store and to Every Cure's existing Postgres instance for the online store. The offline store holds the full feature history; the online store holds a curated subset, materialized on a schedule, that supports interactive queries. Instead of re-stitching base files together to answer a question about a single pair, the team can query the online layer directly and get the latest values back in one call.

To connect the two systems, the team built a custom Kedro dataset for Feast, letting pipelines write to the feature store straight through the Kedro catalog. This integration works through a custom FeastDataset, which is where Kedro's extensibility becomes concrete: by implementing Kedro's dataset interface, the team taught the **Data Catalog** to write to Feast without touching any pipeline code. In a less structured framework, the same outcome would have required significant custom wiring — with Kedro, it required implementing a single, well-defined interface. Reading works differently than a typical Kedro dataset: rather than returning data on load, it hands back a queryable handle, giving the caller control over which entities and features to pull.

The feature lifecycle in the new setup runs in four steps: register, write, materialize, and read.

REGISTER *Entities & features*	WRITE *From Kedro*	MATERIALIZE *Offline → online*	READ *Serve online*
Entities and features are declared once in code. A `FeatureView` ties features to an entity and a source, with typed, documented fields Feast validates writes against.	A custom FeastDataset lets Kedro pipelines write features to the store through the Data Catalog, without touching any pipeline code	A `FeatureService` bundles feature views for online exposure. feast apply registers changes; `feast materialize-incremental` copies offline data to the online store, moving only what changed since the last run.	Applications call `get_online_features` to fetch the latest values for requested entities in a single request, no manual joins or file wrangling across feature files.

A `FeatureService` gives Every Cure a single control point, adding a feature view to the service exposes it to reviewers in the internal application with no other code changes needed. Because materialize-incremental tracks the last materialized timestamp per view, the online store is effectively ephemeral: a single materialize command can rebuild it from scratch, which simplified disaster recovery considerably.



The Role of Open Source

- Kedro is a mature, open source pipeline framework, originally developed at QuantumBlack and now a project of the LF AI & Data Foundation.
- Kedro brings modularity, reproducibility, and separation of concerns to data and AI pipelines — the same design principles that made it straightforward to extend with a custom Feast-writing dataset rather than bespoke integration code.
- Feast is open source, doesn't lock the team into a single cloud platform, and carries relatively low operational complexity compared with other feature stores.
- Feast has no server component of its own. It delegates to BigQuery for offline storage and to Every Cure's existing Postgres for online storage, reusing infrastructure already in place.

The broader result follows from Kedro's design principles: because pipelines are modular and each stage is independently testable and auditable, the team can reliably produce and version features at scale. Because the data catalog abstracts how data is read and written, swapping or extending storage backends is a configuration decision rather than a code change. Teams share and reuse features rather than recomputing them, and every run is reproducible.

Where They Landed

Putting Feast behind Kedro moved feature management away from individual, opaque datasets scattered across the catalog and toward a shared, well-defined registry with explicit schemas and semantics. That registry encourages feature re-use across teams and cuts down on repeated recomputation of the same features.

The setup now serves both of Every Cure's use cases at once: analytical access through BigQuery, and low-latency access through Postgres via the online store. The online store's state is managed entirely by Feast and acts as the source of truth for both the internal application and agentic workflows.

Next Steps

Publishing the registry
Feast keeps the state produced by apply and materialize in a **registry file** used by both stores. By default that file is stored locally; a production setup should manage it remotely, for example in **Google Cloud Storage** or a **SQL** backend.

Automating apply and materialize
Running feast apply locally doesn't scale well once multiple teams are involved. Every Cure hosts a single monorepository and has automated apply as part of its pull-request merge process, while materialize runs on a daily CRON job.

Customizing the engine
Feast uses a local **compute engine** by default, which is enough for smaller datasets. Larger workloads can move to a distributed compute engine such as **Spark**, which handles materialization jobs among other tasks.

> Putting Feast behind Kedro turned feature management from a set of opaque, individual datasets into a shared, well-defined registry, giving both the application and its agentic workflows one consistent source of truth to read from.

Lessons Learned

The most important lesson is about sequencing. Establishing Kedro as the pipeline framework first is what made the Feast integration tractable. The custom `FeastDataset` works because Kedro's dataset interface gave the team a clear, well-defined extension point to build against. Teams that invest in pipeline engineering discipline early reduce the cost of integrating new tools later — not because the tools become simpler, but because the framework absorbs the complexity. The Feast integration also confirmed that Kedro's extensibility is practical, not just theoretical. Adding a net-new storage backend required implementing one interface, with no changes to existing pipeline code.

Get Started with Kedro and Feast

Read the full write-up on the Kedro blog, explore the custom `FeastDataset` implementation, or dig into Feast's own documentation for registries and compute engines. Kedro and Feast are both projects of the LF AI & Data Foundation.

- Blog post: **Building a Feature Store with Kedro and Feast**
- `FeastDataset`: kedro-plugins on **GitHub**
- Feast documentation: **docs.feast.dev**
- Kedro: **kedro.org**