ENGINEERING

Upload Anything: How We Revolutionized Data Upload

CONOR LANDRY , ENGINEER • TUESDAY, OCTOBER 25, 2022

We launched Upload Anything to enable our users to bring any file format into Felt and see it magically "just work." Here's why it's revolutionary.

Making a map has a startup cost unlike any other medium we typically work with (e.g. docs, slides, spreadsheets). When you go to make a map, you have to assemble the datasets you want to visualize and upload them into your software. At Felt, we want our users to grab any file on hand, drag and drop it into the browser and be on their way to making a great map.

So we released 'Upload Anything': one upload button that is smart enough to make most decisions for you. Within seconds, anyone can upload complex geospatial datasets (or images!) and see them on a fast, responsive map (built with Protomaps), and those data layers can be styled in the Felt UI. The result feels buttery smooth and unlike anything else that currently exists for map-making.

To achieve this level of simplicity for the user was technically challenging, required a feat of engineering and some key architectural decisions. We had to build an architecture that accounted for:

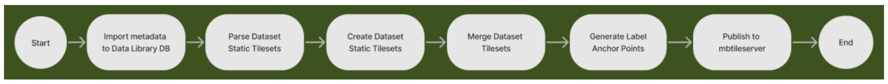
- **Pipeline speed optimization** - how do we execute several computations to visualize the data accurately from the cloud while remaining fast, so uploading anything feels magical to the user no matter their file size?
- **Increased File type & size support** - how can Felt quickly identify any file type and know what to do with it? How can we support very large files that are needed for sophisticated mapping use cases?
- **Ease of use** - How do we do this in a way that feels fantastic to use in the product?

The decisions we made not only helped us achieve these outcomes, but also provided some key benefits that will make allow us to ship more quickly and confidently. Here's how we did it.

Upload Anything 1.0

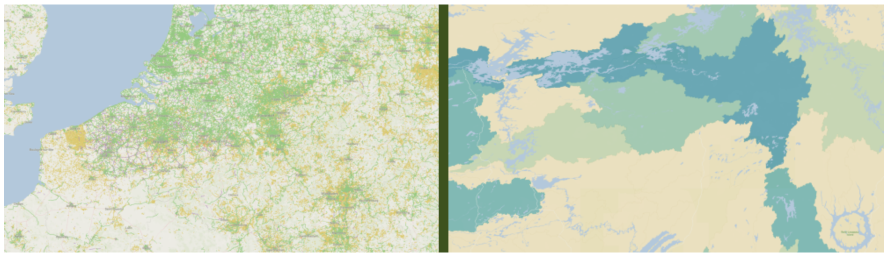
Upload Anything started as an internal tool we built to prove that there's an easier way to make maps. Felt's multiplayer and collaborative features are built with Elixir and while Elixir has served us well for our multiplayer backend, Python has been the lingua franca for data scientists and especially GIS professionals for decades. We wanted to stand on the shoulders of giants and reuse many of the great tooling available to us.

Our initial design was simple; FastAPI provided a thin control layer for kicking off the processing of datasets in GeoJSON or Shapefile format. Datasets were primarily processed using a combination of ogr2ogr, Tippecanoe, PostGIS, and Shapely through a series of Celery pipelines. Once processing finished, we served metadata about the tileset through the same API and the tileset itself through mbtileserver.


Start
Import metadata to Data Library DB
Parse Dataset Static Tilesets
Create Dataset Static Tilesets
Merge Dataset Tilesets
Generate Label Anchor Points
Publish to mbtileserver
End

We needed a way to let our team interact with the pipeline API, so we built an app with Retool that allowed us to upload datasets and publish them to Felt's layer library.

Our cartography team assembled over 50 handcrafted layers for our public launch with this stack. These include a few personal favorites of mine, like the global Bike Lanes layer and the Riverine Flood risk layer.



Our team had successfully proven that we were on to a new, better method of uploading and styling spatial data. However, using this pipeline and API architecture came with a lot of frustration. The pipeline is slow and didn't take advantage of throwing concurrency at an embarrassingly parallel problem. Long-running Celery jobs were canceled during deploys, causing restarts of datasets that had already been processed for more than 6 hours and slowing down our shipping rate. When a dataset did finish processing, it was a game of chance as to whether our Tippecanoe configuration was correct and whether we ended up with an oversimplified blob of polygons depending on the zoom level. Worst of all, we only supported a handful of data formats. To achieve our goals, we would need to go back to the drawing board.

Upload Anything 2.0

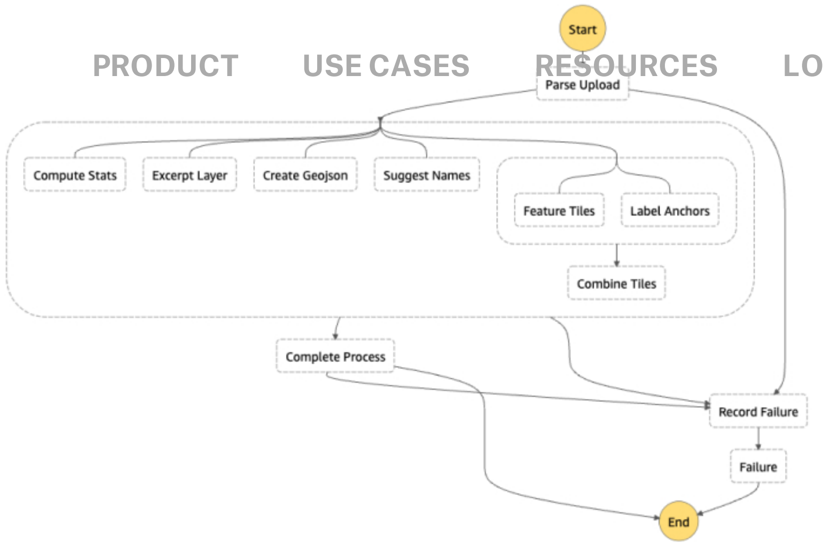
We took our learnings and returned to two key challenges. We needed to build an architecture that would:

- Support every geospatial dataset imaginable, in a way that was quick and accurate.
- Allow for easy internal development and experimentation, and for fast and frequent shipping to production (i.e. It had to be fast to process maps, quick to serve maps, and easy for everyone to use).

All In on AWS

To achieve these outcomes, we chose a risky-but-feasible path. We went "Full AWS" and used Amazon's many primitives & services to assemble a flexible data pipeline with minimal maintenance overhead. This lets us discard many of our explicit components like queues, state management, pipeline execution, and wrangling containers in favor of implicit features of AWS services like Step Functions, Lambas, and Elastic Container Service (ECS).

The job of the pipeline is straightforward: a request is made to produce tiles, then a one-time, signed AWS S3 upload URL is provided back to the Elixir application. The user uploads their file directly to an s3 bucket, kicking off an AWS Step Function definition, which orchestrates a series of concurrent execution steps.



After the user's upload finishes, it's parsed into an OGR-compatible data source. We decided to use GeoJSONseq because it's an easy-to-understand specification and allows us to chew through massive datasets without allocating enough memory to read the whole dataset.

Next, we took advantage of the once-passed-over embarrassingly parallel problem from earlier and split up our work across several concurrently executing Lambda functions, including:

- **Compute Stats** - to extract aggregate statistics about attributes in the dataset
- **Excerpt Layer** - to allow for a table view of the data
- **Feature Tiles** - to convert to scale-independent vector map tiles
- **Anchor Tiles** - heuristically decided single points for map label placement

This new architecture also includes a stable contract for communication between the pipeline and app, ensuring we can ship safely. During the execution of our Step Function, each Lambda incrementally updates an API response for the progress made while processing the dataset. Our Elixir app checks this progress periodically using an Oban Job and gives users real-time progress updates using Phoenix Channels. Since there's a stable contract for communication between the Elixir App and Pipeline, we can easily make changes without breaking Felt so long as the pipeline always returns data conforming to the API definition.

Swapping for speed: from mbtiles to pmtiles

After the pipeline finishes, we need to provide the freshly created vector tilesets to the user. In our first iteration, we served tilesets from a single mbtilesserver instance. Given the size a single tile can reach, this led to poor performance for users on Felt in different parts of the world. Instead, we adopted pmtiles, a single-file archive format for tiled data. Doing so lets us serve tilesets through a CDN, providing low-latency tile reads for everyone.

Key Outcomes

The user experience of this work speaks for itself when you go upload a file–but here's what you may not see:

- **No internal workflow disruptions** - with every Pull Request, an independent staging environment is created for our pipeline & CDN infrastructure. No more asking if colleagues are using staging or not.
- **Increased feature development flexibility** - when we realized Lambda functions have a 15-minute execution limit, we were able to quickly modify the Step Function design to automatically cut over to ECS instances instead, resulting in the ability for users to upload massive datasets (up to 5 GB); no idle server management necessary. This ease was directly related to building the entirety of our pipeline on AWS.
- **Smoother user experience** - When it's time to deploy Felt, executing pipelines continue running the old Step Function code, and any new jobs execute using the new Step Function code. This prevents us from restarting long-running jobs and disrupting our user's workflows.

If you haven't yet, **try it for free**. Let us know what you think at @felt.

Careers

We are actively looking for Elixir Engineers, a Growth Marketer, and an Engineering Manager. If you're inspired by our work, apply to join the team.

More articles

Felt Reaches 1.0, Now Ready for Teams

SAM HASHEMI , CEO • FEBRUARY 28, 2023

Felt Renders Faster Than Ever with MapLibre

CHRIS LOER , ENGINEER • FEBRUARY 28, 2023

January Spotlight: 10 Best Felt Community Maps

ANNA SAVINA , CONTENT AND COMMUNITY • FEBRUARY 7, 2023

Beautiful New Ways to Visualize Data in Felt

MAMATA AKELLA AND ISAAC BESORA , DATA TEAM • FEBRUARY 1, 2023

New in Felt: Geocoding, Geomatching, and Choropleths

MICHAL MIGURSKI , DATA PRODUCTS • JANUARY 24, 2023

Creating UI Delight: Dynamically Rotating Mouse Cursors

TOM HICKS , ENGINEER • JANUARY 10, 2023

37° 48' 15.7068" N, 122° 16' 15.9996" W MODERN MAP MAKING HAS ARRIVED 37° 48' 15.7068" N, 122° 16' 15.9996"

PRODUCT USE CASES RESOURCES LOG IN SIGN UP

The best way to work together on maps on the internet.

SIGN UP

PRODUCT	USE CASES	RESOURCES	COMPANY
CREATE A MAP	PLANNING	BLOG	ABOUT
VISUALIZE DATA	TECH	HELP CENTER	CAREERS
MAP GALLERY	CONSULTING	COMMUNITY	TERMS
	DISASTERS	TWITTER	PRIVACY
	UTILITIES	MASTODON	MEDIA KIT
	EDUCATION	LINKEDIN	
	FOR EVERYONE	FACEBOOK	
		YOUTUBE	