Rebuilding the Plane While It's Still Flying

Tyler A. Young
Felt

It's Still Flying: Large-Scale, Zero-Downtime Migrations Without Fear

Tyler A. Young
Felt

Hi, I’m Tyler.

- Soft real-time & distributed systems
- Writing Elixir since 2019, when I built an MMO server for the X-Plane flight sim
- Working at Felt, “the best way to work with maps, together”

Part 1

Setting the Stage

What’s a migration?

- Moving from one version of a system, architecture, database structure, etc. to a new one
- Triggered by growth, new systems, or changing requirements

“Software engineering is programming integrated over time.”

Titus Winters, coauthor of
Software Engineering at Google

Types of migrations

- Database schema
- Data
- Code
- Whole (sub-)systems

Why would migrations be hard?

- They’re not! …unless:
  - They’ll be hard to undo
  - They can’t be done atomically
  - They need to be done without downtime
  - Other feature dev continues apace
- Overlooked, underestimated

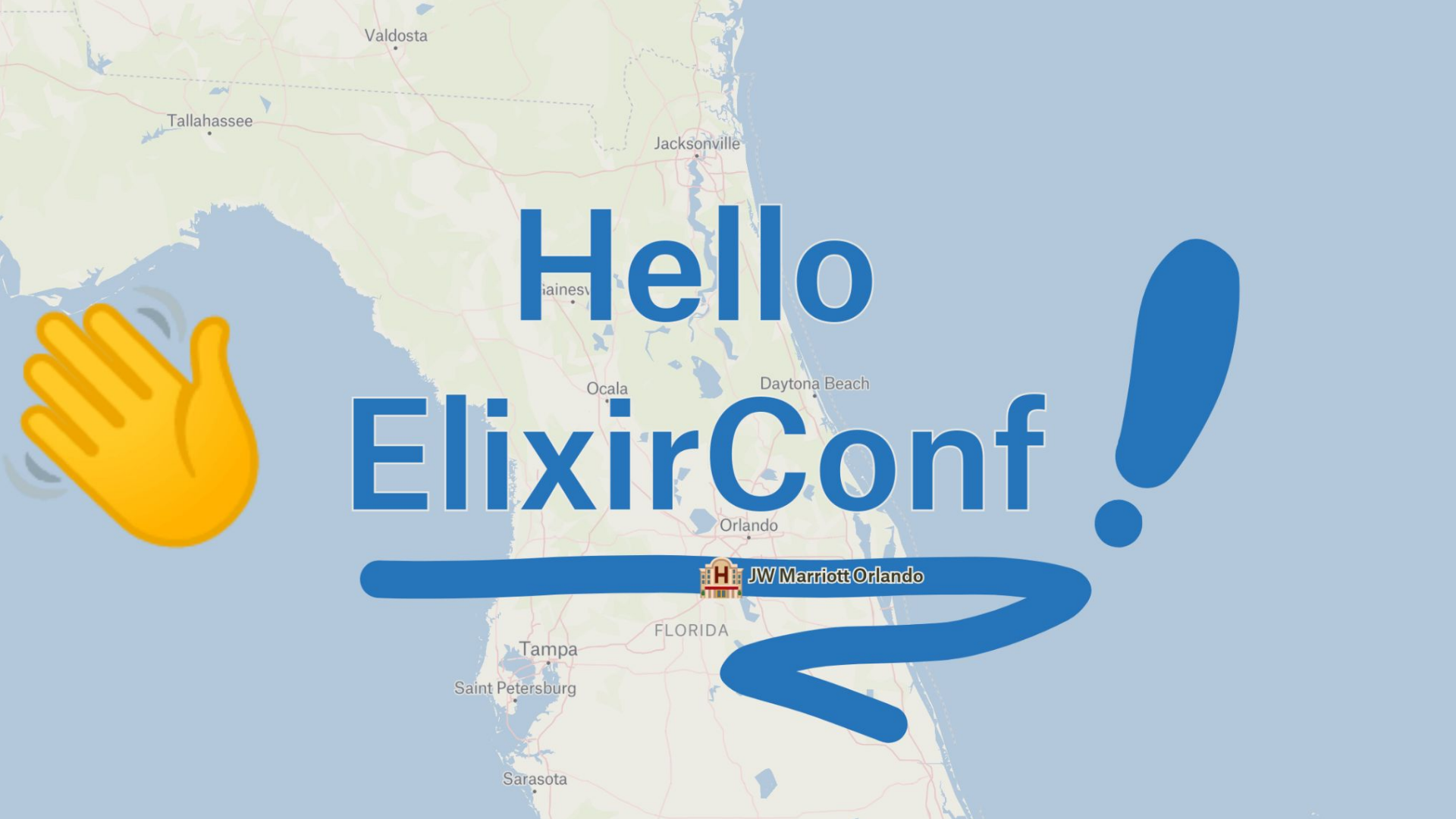

IT'S ONE MIGRATION, MICHAEL.
HOW LONG COULD IT TAKE, 10 MINUTES?
imgflip.com

How we run database migrations

- Blue-green deploys
- Schema migrations run before the new server boots
  - While the prior deploy is running!
- Data migrations triggered manually (and asynchronously) afterward

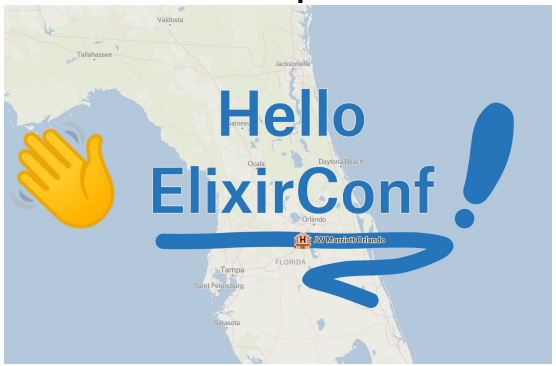
Part 2

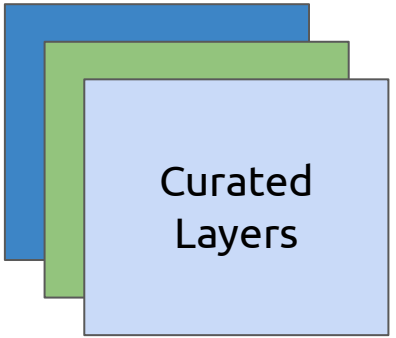
Oral History


Valdosta
Tallahassee
Jacksonville
Hello
Ocala
Daytona Beach
ElixirConf!
Orlando
JW Marriott Orlando
FLORIDA
Tampa
Saint Petersburg
Sarasota


Hello
ElixirConf
Valdosta
Cumberland Island Wilderness
Tallahassee
Big Gum Swamp Wilderness
Jacksonville
Bradwell Bay Wilderness
Juniper Prairie Wilderness
Daytona Beach
Orlando
JW Marriott Orlando
FLORIDA
Tampa
Saint Petersburg
Sarasota
Wildfires, 2021
Burned Areas
Wilderness Areas
Intact Forest Landscapes
2020 IFL Extent
2000 IFL Extent
Forest Zone Extent
Riverine Flood Risk
Extremely High
High
Medium - High
Low - Medium
Low

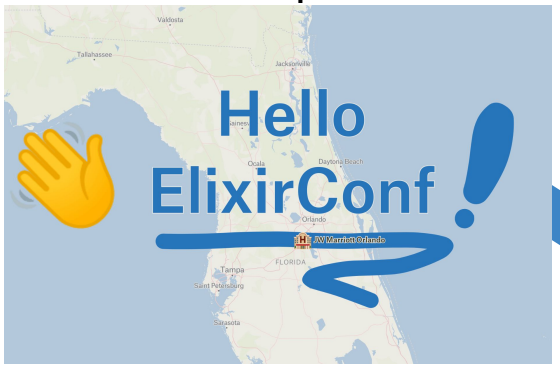
Map





Hello ElixirConf

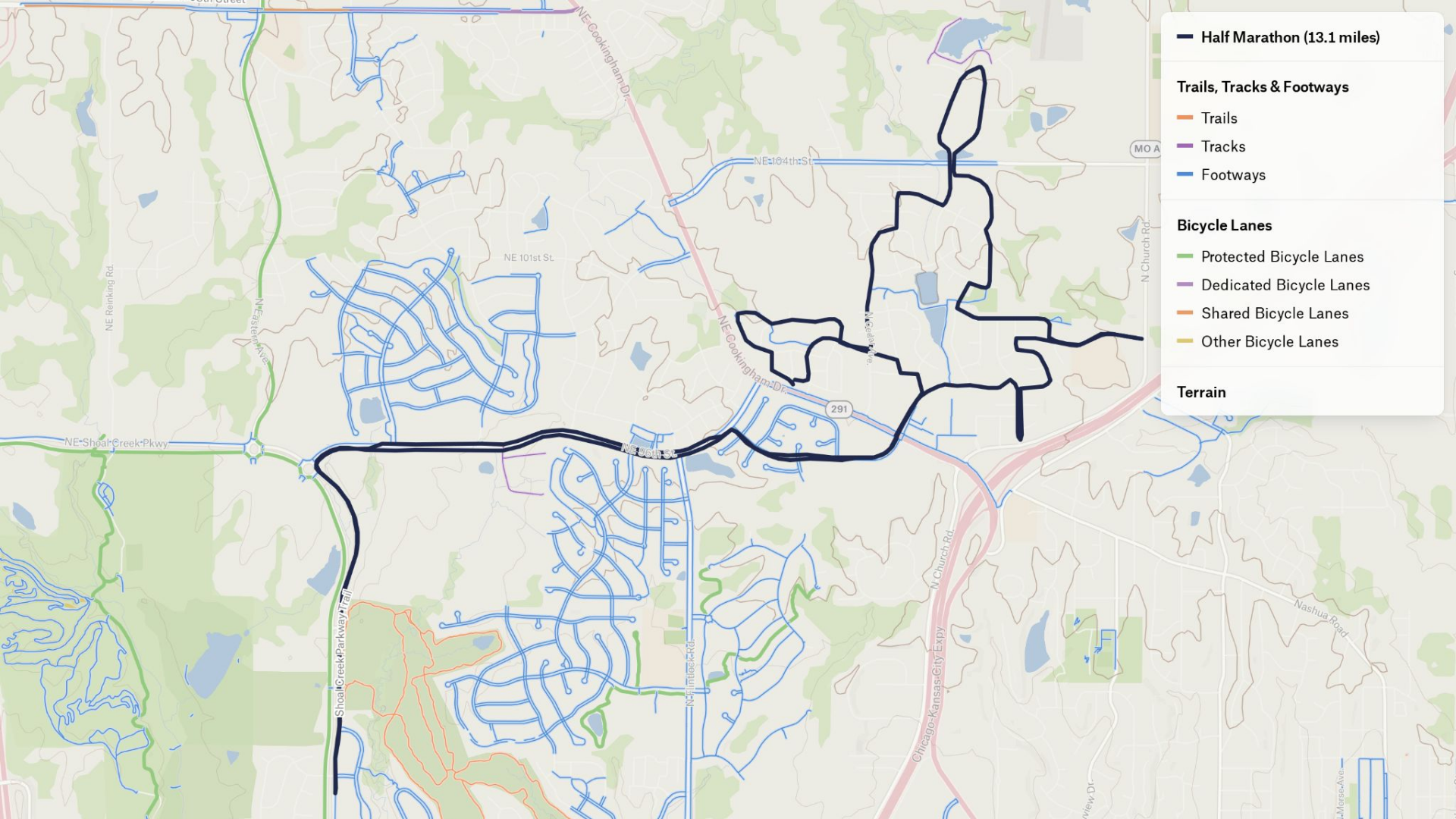
Map



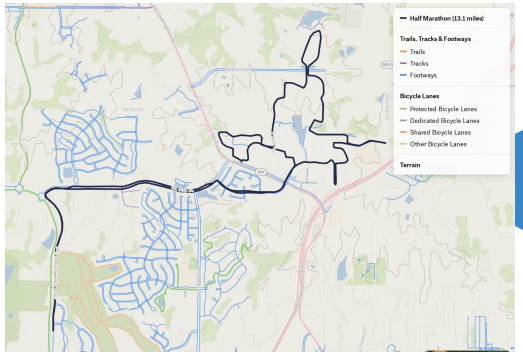
MapLayer

Curated Layers

“How do I create my own layers?”


Half Marathon (13.1 miles)
Trails, Tracks & Footways
Trails
Tracks
Footways
Bicycle Lanes
Protected Bicycle Lanes
Dedicated Bicycle Lanes
Shared Bicycle Lanes
Other Bicycle Lanes
Terrain
NE Cookingham Dr.
NE 104th St
NE 101st St
NE Reinking Rd.
NE Eastern Ave
NE Shoal Creek Pkwy
NE 98th St
Shoal Creek Parkway Trail
NE Timlick Rd
N Church Rd
Chicago-Kansas City Expy
Nashua Road
291
MO A

Map



MapLayer

UserLayer

MapLayer

Curated Layers


USERS
USER LAYERS
CURATED LAYER FEATURES
imgflip.com

The dream

A single, unified abstraction

The dream

Maximally powerful tools for our users

The dream

~~Maximally powerful tools for our users~~

The dream

A single, unified abstraction

It gets worse before it gets better

We need a third layer type.

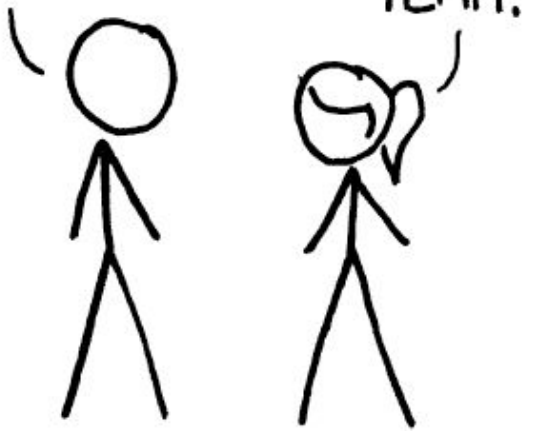
The one, true Layer.

One Layer to rule them all.

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC.)

SITUATION:
THERE ARE
14 COMPETING
STANDARDS.

14?! RIDICULOUS!
WE NEED TO DEVELOP
ONE UNIVERSAL STANDARD
THAT COVERS EVERYONE'S
USE CASES.



SOON:

SITUATION:
THERE ARE
15 COMPETING
STANDARDS.

Hard requirements

1. No downtime
1. No breaking past user data

Getting from here to there

Baby steps!

1. Pay off old tech debt

“For each desired change, make the change easy (warning: this may be hard), then make the easy change.”

Kent Beck, cosigner of
The Agile Manifesto, creator of
Extreme Programming

Getting from here to there

Baby steps!

1. Pay off old tech debt
2. New abstraction to live alongside the old ones

Getting from here to there

Baby steps!

1. Pay off old tech debt
2. New abstraction to live alongside the old ones
3. Compatibility shim to translate between old & new

Getting from here to there

Operation Unbreak the App

Baby steps!

1. Pay off old tech debt
2. New abstraction to live alongside the old ones
3. Compatibility shim to translate between old & new

Getting from here to there

Baby steps!

1. Pay off old tech debt
2. New abstraction to live alongside the old ones
3. Compatibility shim to translate between old & new
4. Uncouple the frontend from the DB schema

The scary final step

- Migrate all user data to the new structure
  - Runs asynchronously
- What happens if a user is editing the old structure while the migration happens?
- What if we need to roll it back?

Silence is golden

1. Back up the database!
2. Run the migration!
3. Wait…

Part 3

Lessons Learned

Feature flags are your friend

- Opt-in to the new (broken?) version of the feature
  - Just the devs working on the project
  - All Felt employees
  - All users
- FunWithFlags library by Tommaso Pavese

Use Excellent Migrations

- Credo check by Artur Sulej
- Automatically checks schema migrations for things that might lock the database
- Deeply understand before ignoring a rule
    - De-risk future copypasta! Leave a comment explaining why ignoring was okay

Lean on Safe Ecto Migrations

- David Bernheisel's series on Fly.io's "Phoenix Files"
- The Ecto migration bible
- Whatever pattern your migration uses, it should probably link back to the relevant section of this series as justification that it's safe!

Small, incremental steps

- The more “save points” the better
  - Corollary: 💜 trunk-based development
- You don’t have to know every step along the way
- Test plan: “Before this change, *x* is true; after, *y* should be true.”

Practice like you play

- Build the habit of making every migration zero downtime, zero data loss way before it really matters
- Slows down dev in the short term, but builds skills you can’t acquire elsewhere
- Smaller-scale changes help bring all the team up to speed on our practices & tooling around this stuff

“You know something if you have experience using it.”

Brian Marick summarizing *Communities of Practice* by Etienne Wenger

There's no substitute for prod data

- Run data migrations against a database dump from prod
  - "Why is this so slow?"
  - "How did 4 of 100,000 rows get a null value there?"
  - "Why does one row have status: success, but no data?"
- Failure to do this means having to make multiple attempts (or worse!)

A microframework for backfills

- Based on the recipe for “Batching Deterministic Data” backfills in David Bernheisel’s *Safe Ecto Migrations*
- You supply:
  - the base query
  - batch size
  - sleep time
  - an open-ended `migrate/1` that operates on the results

```elixir
defmodule MyApp.Repo.Migrations.BackfillOnboardingStep do
  use DataMigration

  @impl DataMigration
  def base_query do
    from(u in "users",
      where: is_nil(u.onboarding_step),
      select: %{id: u.id}
    )
  end

  @impl DataMigration
  def config do
    %DataMigration.Config{batch_size: 100, throttle_ms: 1_000}
  end

```

tylerayoung.com/2023/08/13/migrations/

```elixir
  end

  @impl DataMigration
  def config do
    %DataMigration.Config{batch_size: 100, throttle_ms: 1_000}
  end

  @impl DataMigration
  def migrate(results) do
    Enum.each(results, fn %{id: user_id} ->
      user_id
      |> Ecto.UUID.cast!()
      |> MyApp.Accounts.set_onboarding_step!()
    end)
  end
end
```

tylerayoung.com/2023/08/13/migrations/

More resources

- “Safe Ecto Migrations” by David Bernheisel
  fly.io/phoenix-files/safe-ecto-migrations/
- “Migrations Done Well” by Gergely Orosz
  newsletter.pragmaticengineer.com/p/migrations
- Excellent Migrations Credo check by Artur Sulej
  github.com/Artur-Sulej/excellent_migrations
- FunWithFlags by Tommaso Pavese
  github.com/tompave/fun_with_flags