MEWS

Automated zero-downtime migrations

Josef Starýchfojtů
@JStarychfojtu

Problem

- Downtime during migrations
- Deploying often (makes it a real problem)
- Operating over the whole globe
- Annoying migrations for developers, which tends to legacy schema

Solution ?

- Just don’t make breaking changes !
- Example: Column remove
  - 1) Stop using the column
  - 2) Remove it

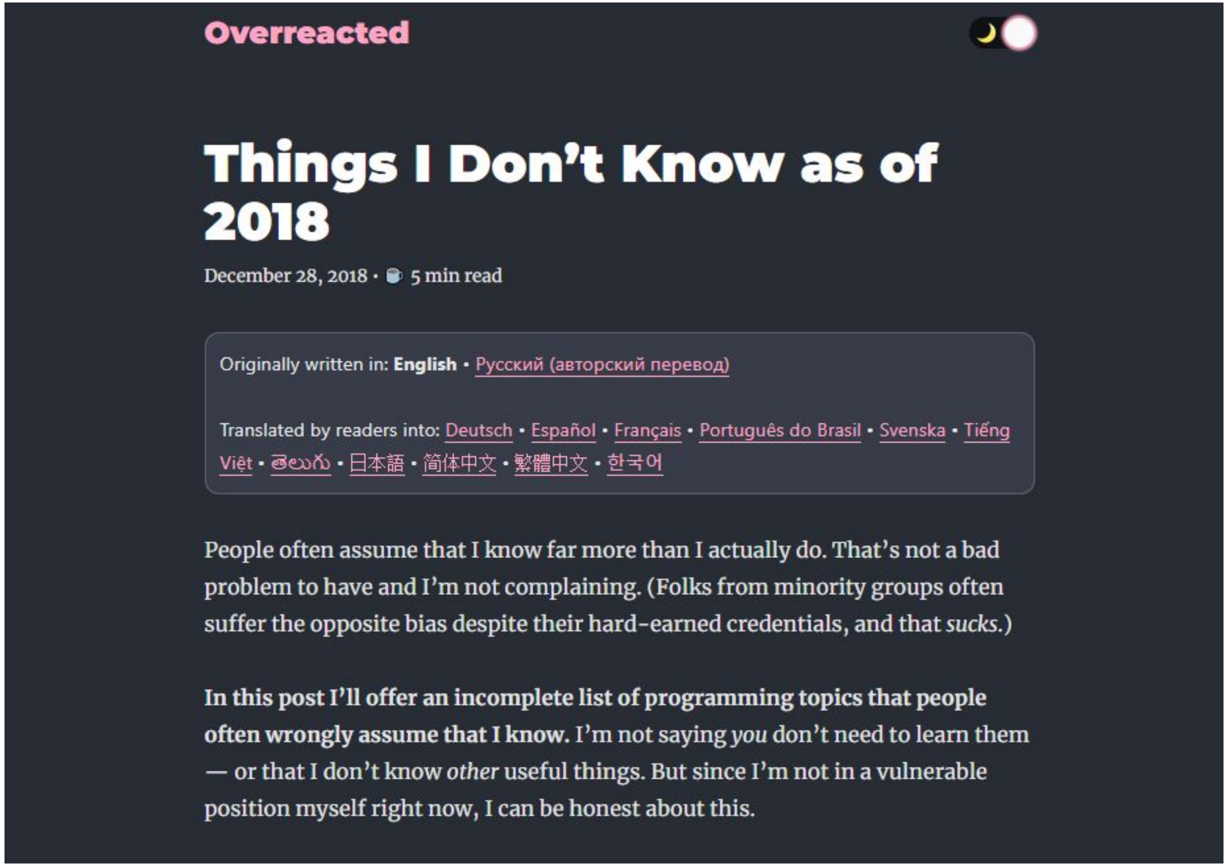
Problem with the solution

- How to control what are breaking changes ?
- How to control what will cause long locks and thus causes downtime anyway ?
- How to make migrations less imperative ?

Let’s dive into the idea and our solution

Iterative and custom nature

- We write it on demand, we learn as we go
- The specific implementation is suited on our use-cases
- Some edge cases are not resolved, especially for large tables, where custom solution is still required
- Think of it rather as minimizing downtime to very rare edge cases (no downtime was caused so far)



Overreacted

Things I Don't Know as of 2018

December 28, 2018 • ☕ 5 min read

Originally written in: **English** • Русский (авторский перевод)

Translated by readers into: Deutsch • Español • Français • Português do Brasil • Svenska • Tiếng Việt • తెలుగు • 日本語 • 简体中文 • 繁體中文 • 한국어

People often assume that I know far more than I actually do. That's not a bad problem to have and I'm not complaining. (Folks from minority groups often suffer the opposite bias despite their hard-earned credentials, and that *sucks*.)

In this post I'll offer an incomplete list of programming topics that people often wrongly assume that I know. I'm not saying *you* don't need to learn them — or that I don't know *other* useful things. But since I'm not in a vulnerable position myself right now, I can be honest about this.

https://overreacted.io/things-i-dont-know-as-of-2018/

Entity Framework

Entity
Framework

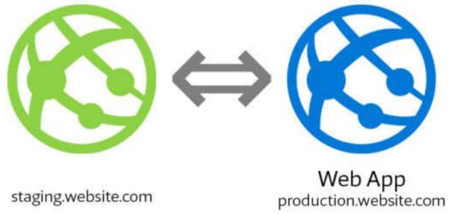
- SQL Server
- Schema defined in code by classes
- EF holds serialized schema in its own table
- During initialization, it compares the schema with the current state in code and generates a diff
- Diff is not exposed directly, one has to inherit SqlGenerator

Our environment

- Azure
- 2 regions, 3 app services (WEU web, WEU jobs, NEU web)
- 2 slots (Live and Staging)
- Live serves traffic
- Staging is used for deploy
- 15 instances

Our deploy

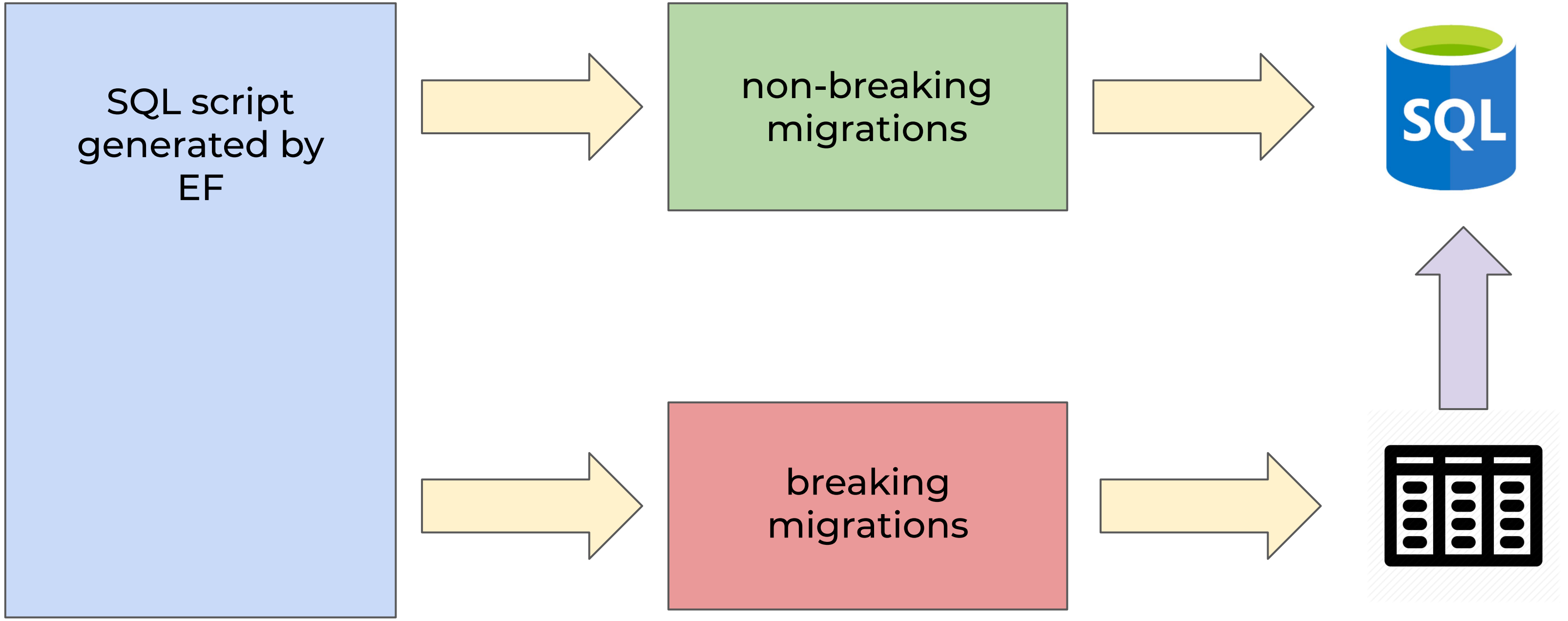
- Staging is booted up with live configuration
- One of the instances locks the platform info and starts migrating (other are waiting)
- Here mainly the migrations take place (with old version still running in Live slot)
- All instances are heated up
- Traffic is swapped to the new slot



EF automated migrations

- EF gives us a set of operations (diff of schema and classes)
- Based on that we generate 2 sets of migrations
  - Non-breaking, executed right away, compatible with both versions of app
  - Breaking, executed when all instances are running on the new version, incompatible with old version

EF automated migrations



Example: Drop column

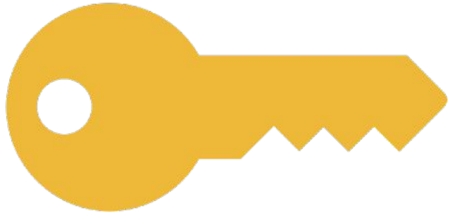
- Check that the column is nullable, report error otherwise
- Breaking:
  - Drop the column with its indexes

Example: Add column

- Check that given type can be added online (e.g. rowversion)
- Check that the column is not generated by EF
- Non-breaking:
  - Add the column

Example: Add foreign key

- Check that there is an index defined (performance of the migration)
- Breaking:
  - Add the foreign key



Example: Alter column

- Check that alter to NOT NULL is done only on small tables
- Migration is dependent on the change:
  - Non-breaking if the change is to nullable (ideally, the value still has to be treated as not nullable in the new version, in order not to break old version with null value, this could be made a breaking migration for that reason to enforce it)
  - Breaking if the change it to not nullable
  - Precision changes not resolved yet

Example: Rename column

- Done via 2 columns (old and new)
- Nonbreaking:
    - Make the old column nullable
    - Add a new column with indexes of the old one
    - Drop the foreign key from old column (optional)
    - Create a trigger to synchronize both columns on insert/update
    - Create a migration to transfer all data from old column to the new one

Example: Rename column (2)

- Breaking:
  - Drop the old column with its indexes
  - Remove triggers
  - Add foreign key on the new column (optional)
  - Make the new column not nullable (optional)

What we achieved ?

- Migrations are declarative
- Migrations automated and seamless = take much less time
- Migrations are controlled again past mistakes (performance issues, locks)
- Naming is kept up to date (since it is not hard to rename)
- It is about 500 lines of code, go and write it yourself !

Thank you for your attention!