Engineer

Migration

a planned move to a new version or platform

What it is

A migration is a planned move of something already in use to a new version or platform, carried out in steps, with the old one retired at the end. The thing being moved can be a whole workload, an application, a body of data or the structure that data is kept in. What separates a migration from a rebuild is that the system stays in use while it moves: its users, records and integrations come along, and the old version keeps working until the new one has taken over.

Engineers use the word at two scales, often without saying which one they mean.

At the large scale it is a move to a new platform or a new major version. Microsoft's Cloud Adoption Framework treats moving workloads to Azure, from servers on premises or from another cloud, as five steps: plan, prepare the workloads, execute, optimise, and decommission the source. Esri's customers move their maps and data from ArcMap to ArcGIS Pro. A Next.js application moves from the Pages Router to the App Router, and `getServerSideProps` gives way to fetching data inside Server Components. A PostgreSQL cluster moves to a new major version, which often changes the layout of the system tables, so the data is carried across with `pg_upgrade` or with a dump and restore.

At the small scale a migration is one versioned change to a database's structure, kept as a file in the repository and applied by a tool in a fixed order. Django calls its migrations a version control system for the database schema: each file records a change to the models, is committed with the code, and runs in the same sequence on a developer's laptop, on staging and in production. GitLab ships batched background migrations that rewrite large tables a batch at a time after

an upgrade, for example widening integer columns to `bigint` before they overflow. A data migration belongs here too: one that changes the rows rather than the columns, such as filling a new field from two old ones.

These are one idea at two scales. Each has a known starting state, an ordered set of steps, a target, and a way back if a step goes wrong. A platform move usually contains schema migrations, and a long run of schema migrations is, over the years, a platform changing piece by piece under the people who use it.

Use the concept in a decision

Put “What is a migration?” to work in a real decision. We can help relate the concept to the system you are building or buying. Data taken out of something ageing and into something supported, with the business still running while it happens.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Improved GIS map performance 10x on PostgreSQL — 10x faster maps, after moving off MSSQL

Improved GIS map performance 10x on PostgreSQL Get in touch →

Why it matters

Software outlives what it was built on. Cloud contracts are renegotiated, frameworks change their model, and vendors retire products. A system kept for a decade will be migrated several times in that decade, so how well a team migrates ends up mattering more than how well it chose the first platform.

- **The split state is where the cost lands.** While part of a system runs on the old platform and part on the new, both have to be operated, monitored and kept consistent. The Cloud Adoption Framework tells teams to keep that period short and to move components that depend directly on each other in the same wave.
- **A database cannot be redeployed.** Code can be built again and replaced; the rows in production cannot. Schema migrations turn a change to live structure into a file that can be reviewed, tested against a copy, applied in order and, for most changes, reversed. The alternative is a production schema that is whatever the last person with a console left behind.
- **A planned move can be undone.** The Cloud Adoption Framework asks for rollback criteria and procedures to be set before any migration starts, for the rollback to be tested, and for go/no-go checkpoints along the way.

  The trade-offs sit in the cutover. The Cloud Adoption Framework separates migration with downtime from migration with near-zero downtime, where data is replicated continuously and the switch comes at the end. For a major version of

PostgreSQL, `pg_upgrade` shortens the cutover: it skips the dump and restore such an upgrade typically requires by reusing the existing data files, and in `--link` mode, which copies no files, it is much faster again.

What it looks like in practice

Three projects on this site show the term at three sizes, from a whole platform down to the order of the files in one repository.

One migration was a move off a cloud provider. A client's AWS setup had been over-provisioned, and the work began by reviewing what was really used: unnecessary services went, and the remaining capacity was adjusted. The workloads then left AWS. The engagement record supports those infrastructure changes. Cutting unused services before moving meant there was less to carry; it does not establish a like-for-like cost ratio, a productivity result or a disruption-free migration.

The second was an engine change with a new design at the far end. A geographic map application's database moved from MSSQL to PostgreSQL with the PostGIS extension, and the target was not a copy of the old structure. The new schema was designed for spatial data, with GiST and SP-GiST indexes and strict foreign key and check constraints that enforce validation rules on spatial coordinates. Over 50 million spatial records travelled into it through ETL pipelines that transformed them to EPSG:4326. A move onto a new design is the chance to state the data's rules in the schema itself. The page reports a 10x improvement in spatial query performance and backend CPU load down by 65%.

The third is the small scale, kept up for years. One platform's schema is the sum of 1,022 migrations, numbered 036 through 1102, and the page calls the ordering discipline around them unglamorous and non-negotiable. The history in the repository is a description of production, and it stays true only while any database, from a developer's fresh copy to a restore, can be rebuilt by replaying it from the first file to the last.

Where it goes wrong

The cutover gets the planning, and the trouble tends to gather in the weeks around it, in four recognisable shapes.

Two systems run side by side for longer than planned. The move stalls halfway, the pieces left are the hard ones, and the team runs both platforms for months. Users feel the seam: the Next.js guide notes that moving between a Pages Router page and an App Router page is a hard navigation, and that `next/link` does not prefetch across the two. Keep the waves small and make each next step cheap. One frontend here moves its pages onto a shared factory that loads each page in one round trip, so every converted page inherits the behaviour.

A cutover with no way back. Some steps change the source as well as the target. With `pg_upgrade --link`, once the new cluster has started it has written to files the old one shares, and the old cluster then has to be restored from backup. Esri warns that a dataset upgraded for ArcGIS Pro may stop being usable in ArcMap, and tells users to copy it first. In Django, a `RunSQL` operation written without reverse SQL raises `IrreversibleError` the moment someone migrates backwards. Set the rollback criteria before the first step, rehearse the route back, and turn it into a check: one repository here runs a guard engine whose validators include a check that a migration can be rolled back.

A migration that already ran gets edited. Migration tools record which files have been applied and skip them from then on. Change a file after it has reached production, and production keeps the old version while every fresh database gets the new one: two schemas under one history. Leave applied migrations alone and correct them with a new one; a build that replays the history from empty and compares it with a migrated copy catches the drift.

The upgraded database is slow on its first day. `pg_upgrade` carries most planner statistics across, not all, and the planner guesses on the rest until they are rebuilt. Its `--check` mode changes nothing and can run while the old server is still serving, so run it first, and rebuild the missing statistics straight after the upgrade before gathering the rest:

```
pg_upgrade --check -b "$OLD/bin" -B "$NEW/bin" -d "$OLD/data" -D "$NEW/data"
pg_upgrade -b "$OLD/bin" -B "$NEW/bin" -d "$OLD/data" -D "$NEW/data"
vacuumdb --all --analyze-in-stages --missing-stats-only
vacuumdb --all --analyze-only
```

Further reading

- Plan your migration — Microsoft's method for moving workloads to Azure: waves, cutover choices and rollback planning.

- Migrations — Django's guide to versioned schema changes, reversing them and data migrations.
- pg_upgrade — the PostgreSQL upgrade tool, with check mode, link mode, reverting and the statistics steps afterwards.
- How to migrate from Pages to the App Router — moving a Next.js application to the new router one page at a time.
- Migrate existing work — what carries over from ArcMap to ArcGIS Pro, what has to be converted, and what to back up first.
- Check migrations before upgrade — how GitLab's batched background migrations rewrite large tables after an upgrade, and how to see that they have finished.

Glossary

Choose for your actual constraints

Bring a representative dataset, the questions it must answer and the current bottleneck. We can agree how to measure quality, throughput and recovery.

See what it costs Get in touch →

Make the terminology practical

Ask how "What is a migration?" applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/migration/