Expand Contract for Databases and Services

May 18, 2023 · 4 min

I haven't seen Expand-Contract written about in some years, and I think it is a great way of performing database schema migrations without the need for application downtime. I also realised that it also applies to microservices and service-to-service communication in general.

The Easy Example

One of the two examples given is wanting to change how an address is stored in a database. The schema starts off looking like this:

id	name	address
1	Reaktor	Läntinen Rantakatu 15, 20100, Turku, Finland

The requirement is that the schema is changed to look like this:

id	name	street	postcode	town	country
1	Reaktor	Läntinen Rantakatu 15	20100	Turku	Finland

The way you would traditionally achieve this is with a migration:

```
alter table buildings
  add column street text,
  add column postcode text, -- postcodes can start with a 0, so store them as text
  add column town text,
  add column country text

update buildings set
```

```
  street    = split_part(address, ',', 1),
  postcode  = split_part(address, ',', 2),
  town      = split_part(address, ',', 3),
  country   = split_part(address, ',', 4)
where
  address != ""

alter table buildings
  drop column address
```

The problem with doing this is that the software using this table needs to be stopped while the update is happening; if the old version is running, the app will suddenly be trying to query a non-existing column. If the new version is running, it will also be trying to query non-existing columns.

The process has to look like this:

1. stop the old app
2. run the migration
3. start the new app

Step 2 however can be long, especially if there is lots of data. And what happens if you cannot have downtime for your service?

The Expand Contract Way

1. add a new column to the table (nullable)
2. release new software
   - for reads, read both old and new columns; prefer data in new columns if it exists
   - for writes, write to new columns
3. run a script to migrate any remaining data
4. release new software
   - only reads new columns
   - only writes new columns
5. drop the old column

This is more steps than the original method, but it means there is no downtime in your system. Also, if you make step 2 write to both columns, the migration is easily reversible as no data is lost until the fourth step runs. .

What about APIs? Services?

Expand Contract doesn't have to just be about services either. For example, you have two services and have decided that part of service A should be migrated into service B, which has a similar system. The process is broadly similar to the database example above but with service releases instead:

1. Service B's data model is expanded
2. Service A is released:
    - for reads, read both it's own datastore and Service B. Return result from B if available
    - for writes, write to it's own datastore and Service B
3. Run a script/application to migrate the remaining data
4. Release Service A:
    - uses Service B for all operations
5. Drop old data store tables

As you can see, the process is broadly similar to when implementing a database change; the only difference is some coordination with the other service team. The coordination is only to make sure their data model is ready; no need to release anything at the same time, and no downtime in either service is required.

Downsides

This may sound like a silver bullet, but as with all techniques, it has drawbacks.

The primary drawback is the extra steps required. There are multiple releases, and data migrates lazily/on demand. Then there is the extra step of migrating the remaining data, which is an additional effort.

The other drawback is a symptom of the first drawback: time. It takes far longer to do expand-contract than to have a short downtime. Depending on your application, short

downtime might be the better choice to make. For example, a queue processing service which doesn't have a synchronous API would probably be better choosing the downtime, assuming it can catch up with any messages which queue up during the downtime!

database feature flags microservices architecture

NEXT PAGE »
Feature Flags in a CI Pipeline