

Continuous Delivery & Relational Databases

Adam Szmigin – adam@xsco.net
11-Nov-2014



This work is licensed under the Creative Commons Attribution-NoDerivatives 4.0 International License.
To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd/4.0/> or send a letter to
Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Why This Talk?

Motivation:

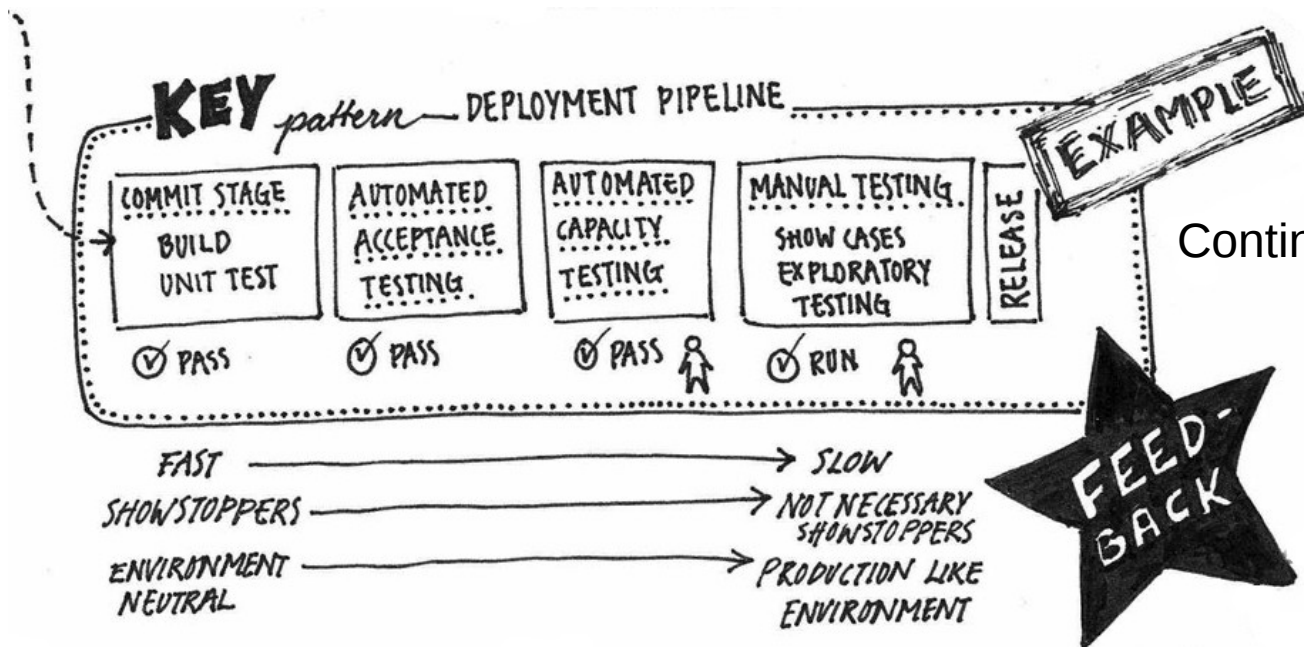
- Promote continuous delivery and automation
- Provide a standard way of working with RDBMS
- Hairy approaches to DB deployment still persist

Inspiration:

- dbdeploy – <http://dbdeploy.com>
- Flyway – <http://flywaydb.org>
- Red Gate – <http://red-gate.com/products/dlm>

Continuous Delivery: Essential Requirements

- Purpose: Continuously deliver new versions of software, measure value to your organisation.
- An automated **deployment pipeline** enables changes to be realised & tested with low effort.



Continuous Delivery Visualisations
by Nhan Ngo

Database Schema Management: Essential Requirements

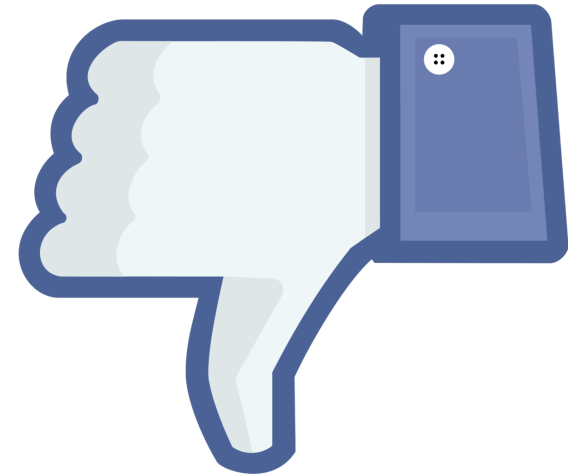
- Non-destructively upgrade an existing database to the next version
- Roll back an existing database to a prior version in the event of a problem
- Create a new database from scratch for testing
- Awareness of what has already been deployed
- Test whether deployment was successful or not
- Manage non-linear development

Typical Approach

- Upgrade handled by scripts written and deployed piecemeal by developers
- Roll-back rarely tested; rely on backup/restore
- Database schema is not under source control; new environments are copied from existing
- Awareness of what has been deployed is not recorded explicitly
- Success of deployment based on visual examination by person running scripts on target environment
- Non-linear development managed by crossing fingers

Commonly-Seen Problems

- Deployment issues past the development phase
 - The order of scripts used for a release to SIT or UAT may not match the order used during development
- Schema of shared development environments in unknown state
- Low confidence in roll-backs
- Feedback too late to developers
- Obstructions to automation



Better Ideas Seen In The Wild

- Numbering scripts to identify strict order
- Full schema under source control; makes it easy to see per-object changes
- Record of which scripts have been run in a dedicated table in the database
- Tools to run certain numbered scripts depending on what is already in a target database

But not quite enough for continuous delivery...

A Gold Standard?



- Source control is the master:
 - Entire database schema
 - All incremental changes (upgrade, rollback)
- Changelog explicitly recorded in the database
 - Full history of changes
- Fully-automated install, upgrade, roll-back
 - Including fully-automated testing
- Full support for non-linear development
 - Recalling that deployment of scripts is always linear

Some Non-Functional Desires...

- Small, lightweight
- Low migration effort
- Works across database technologies
- Works across operating systems
- Natural fit whether writing for JVM, CLR, or native
- Programmatic API for customised integration
- Free (as in beer) software
- Free (as in speech) software

Can something be built?

Can something be built?

Enter `dbmig...`


Semantic Versioning, aka “SemVer”

- Formal, comprehensive approach to versioning
 - See details on <http://semver.org>
 - Major, minor, patch, pre-release, build metadata
- Examples:
 - 3.42.1
 - 12.0.0-alpha
 - 1.3.6-nightly+svn.39275
- Versioning scheme of choice for dbmig:
 - `<Major>.<Minor>.<Patch>+script.<n>`

Source Control is the Master

- Install
 - From no version to a version
- Upgrade
 - From one version to another version
- Latest
 - From nothing to the bleeding edge, no version control (!)

```
MyDb/  
  install/  
    1.2.3/  
      052_inst.sql  
  upgrade/  
    1.3.0/  
      001_foo.sql  
      002_bar.sql  
  latest/  
    ...
```



Version 1.3.0+script.2

Changelog Table in the DB

id	applied	action	from	to	script_hash
1	3-Apr 11:59	install	(null)	1.2.3+script.52	6f3e559bbd...
2	3-Apr 12:04	upgrade	1.2.3+script.52	1.3.0+script.1	4355a46b19...
3	4-Apr 09:38	upgrade	1.3.0+script.1	1.3.0+script.2	53c234e5e8...
4	4-Apr 15:01	rollback	1.3.0+script.2	1.3.0+script.1	53c234e5e8...

- A hash is kept of all scripts
 - Can be used to check whether a script in the repository on disk has changed since deployment

Fully-Automated Deployment

```
$ cd ~/path/to/my/database/repository
```

```
$ dbmig --target=<conn_str> show
```

```
Version installed: 1.3.0+script.1
```

```
$ dbmig --target=<conn_str> migrate
```

```
Run upgrade script 1.3.0/002_bar.sql? [yn] y
```

```
Upgraded to 1.3.0+script.2
```

```
$ dbmig --target=<conn_str> migrate
```

```
  --version=1.2.3+script.52
```

```
Run rollback script 1.3.0/002_bar.sql? [yn] y
```

```
Run rollback script 1.3.0/001_foo.sql? [yn] y
```

```
Rolled back to 1.2.3+script.52
```

Invariants to Test Consistency

- With fully-automated deployment, invariants can be continuously verified by automated build software:

Install Script + Upgrade Scripts
= Latest

Install Script + Upgrade Scripts + Rollback Scripts
= Install Script

Non-Linear Development

- Let's assume an emergency patch release is made and checked into source control...

Developer's local environment:

- Install: 1.2.3+script.52
- Upgrade: 1.3.0+script.1
- Upgrade: 1.3.0+script.2

Continuous build server:

- Install: 1.2.3+script.52
- Upgrade: 1.2.4+script.1
- Upgrade: 1.3.0+script.1
- Upgrade: 1.3.0+script.2

- The conflict can be identified by checking the source code repository against the changelog

Demo time...

dbmig Project Info

- Website: <http://dbmig.xsco.org>
- Written in C++11, using STL, Boost
- Currently tested on PostgreSQL
 - Uses cross-platform DB library SOCI
 - Help wanted to test SQL Server, Oracle
- Currently tested on GNU/Linux
 - Cross-platform code, built using GNU Autotools
 - Help wanted to test Windows, Mac builds

Q & A