Me: David Clark

My Stuff: https://github.com/dwclark

Code Repos: db-reset, flyway-dsl, flyway-dsl-demo

Presentation In: flyway-dsl-demo

Database Migrations

With

Flyway, Gradle, Groovy and Git

In the beginning...

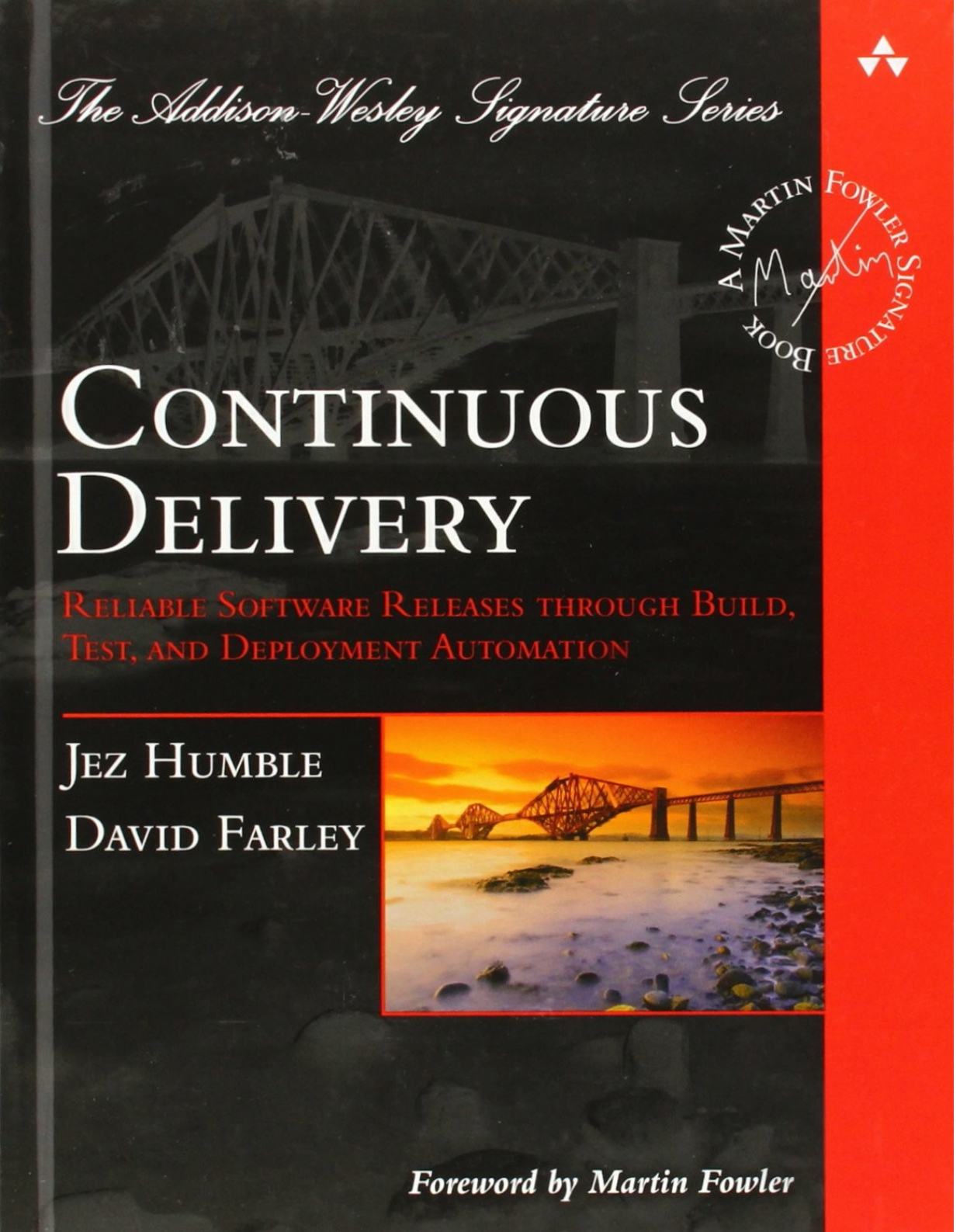
Was the database deployment from hell

Problems

- Deployment day meant everyone showing up with sql scripts to run.
- Scripts would fail with syntax errors or compiler errors.
- Some scripts would clobber what other scripts had done (Oracle packages were a real problem here).
- Scripts would depend on other scripts being run that had not run.
- Scripts would have different requirements for how they were run.
- Scripts were run manually by DBAs, which meant the deployment was non-repeatable in other environments.
- Different environments (dev, qa, uat, and prod) were out of sync with no guaranteed way to get them in sync.
- No way to tell what is in the database, no versions.
- When scripts failed it was hard to tell why they failed.
- Deployments would introduce preventable bugs due to the above. issues, issues that did not exist in other environments.
- DB deployment time: 4-6 hours.

Current State

- Deployment day means we show up with one zip file.
- Have not had a compiler error or syntax error in production.
- Scripts are unlikely to clobber each other.
- Ordering dependencies are gone, there is a strict run order.
- Scripts are always run exactly the same way, in the same context.
- Scripts are run by a tool and are completely repeatable.
- Different environments (dev, qa, uat, and prod) stay in sync and will always eventually converge to a synchronized state.
- Strict database versioning.
- Small scripts lead to easier to diagnose deployment problems.
- Scripts run the same everywhere, if there is a bug in one place it is the same everywhere.
- DB deployment time: 5-20 minutes. Time is directly related to dataset size for DML operations.
- Number of migrations: Release #1: 218 migrations, Release #2: 363 migrations; Plus emergency bug fix releases.



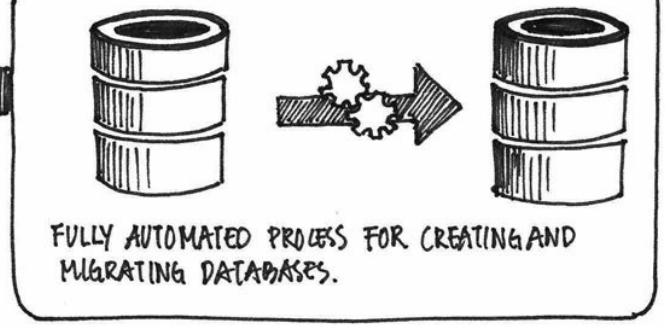
Great Book on Sotware Delivery

Chapter 12 deals with database migrations

What does CONT. DEL. say about

MANAGING DATA

Nhan Ngo



FULLY AUTOMATED PROCESS FOR CREATING AND MIGRATING DATABASES.

- NO REAL DATA BASE
  - BENEFIT : (LAYERS)
    - FOCUS ON BUSINESS BEHAVIOUR
    - +
    - DATA ACCESS CODE KEPT TOGETHER
- IN MEMORY DATABASE
  - CONFIGURABLE (ALLOW YOU TO SWITCH TO ANYTHING SUITABLE)
  - BENEFIT: DECOUPLED CODE

MANAGING TEST DATA
2 CONCERNS
- TEST PERFORMANCE
- TEST ISOLATION

MANAGING THE COUPLING BETWEEN TEST AND DATA

TEST ISOLATION
EACH TEST'S DATA IS ONLY VISIBLE FOR THAT TEST.

ADAPTIVE TEST
EACH TEST IS DESIGNED TO EVALUATE IT'S DATA ENVIRONMENT AND ADAPT IT'S BEHAVIOUR TO SUIT THE DATA IT SEES.

TEST SEQUENCING
TEST ARE DESIGNED TO RUN KNOWN SEQUENCES, EACH DEPENDING FOR INPUTS ON THE OUTPUTS OF IT'S PREDECESSORS.

SETUP & TEAR DOWN

CONSEQUENCE
MORE COMPLEX AND LARGER TESTS.

CONSEQUENCE
FAIL CAUSING SUBSEQUENT TEST NOT TO BE RUN

IF YOU WANT TO TEST DIFFERENT VARIATIONS OF THIS TEST YOU ARE FORCED TO RUN THE PREDECESSORS

COMMIT STAGE → AUTOMATED ACCEPTANCE TEST → CAPACITY TESTING → MANUAL TEST

MUST RUN QUICKLY

- MINIMUM TEST DATA TO ASSERT THAT THE UNIT UNDER TEST EXHIBIT THE EXPECTED RESULT
- TEST NOT CLOSELY TIED TO IMPLEMENTATION. WILL OTHERWISE INHIBIT CHANGE.

+

3 TYPES OF DATA
- TEST SPECIFIC – TEST ISOLATION STRATEGY
- TEST REF. DATA – SUPPORTING CAST
- APPLICATION REF DATA – IRRELEVANT TO BEHAVIOUR UNDER TEST. NEEDS TO BE THERE FOR APPLICATION TO START UP.

+

AMPLIFY TO GET THE LARGE SCALE

SUBSET OF PRODUCTION DATA

Main Idea of Database Migrations

1) Edit the migration scripts
2) Reset the database to a known point
3) Run the migration scripts
4) Test the migration
5) If the migration was not correct, return the step 1

Ideally this should all happen from one tool that can be automated and scripted. Even better if this is a standard tool that developers are already familiar with (or should be familiar with).

DEMO

Webby Webworks

Why not use Liquibase?

- At the time of writing this my exposure to Liquibase was that it was an xml framework built around making vendor neutral migrations.
- It turns out that writing SQL based migrations is now supported, along with XML, YAML, and JSON, but...
- It's not really SQL, it is SQL with comments that have embedded comments that give similar information to XML, JSON, or YAML metadata.
- Putting configuration or executable code in comments is a really bad idea in my opinion (though I reserve the right to do it myself).
- The big one is that like most migration tools, you have to embed all of the code in your migration scripts. For moving targets like Oracle packages or T-SQL stored procedures, this makes source code control for the actual code (the packages and procedures) much more difficult.
- But if Liquibase works for you, by all means use it.

Why not use Rake/Rails?

- You define your migrations in Ruby files. Some using a Ruby DSL and others using SQL.
- I can't emphasize enough how much of a deal breaker this is for a lot of shops. SQL programmers want to store their stuff in sql files. As Java programmers would you accept this: “Well, you can program in Java all you want, you just have to embed all of your Java code as strings inside C files.”
- The concept of forward and backward migrations is interesting but I believe is fatally flawed. The backward migration will almost always involve data loss in a production environment and will almost certainly be not as well tested in a development environment. Database specific tools can give guaranteed rollbacks, use those instead of rolling it in your code.
- We are Java programmers. Will learning and using Rake is not hard, it is all of the support infrastructure that will be a pain. A language is not just syntax, it is an entire infrastructure as well.
- Again, if rake works for you, use it.

TOOLS

Flyway

- Stores all metadata in schema_version table inside your database.
- All scripts are run in order.
- All scripts are run exactly once.
- Version information is stored in the script's file name.
- Scripts are written using the native scripting language or your database (PL/SQL, T-SQL, plpgsql, etc.).
- Uses standard JDBC drivers to access database.
- Can be used from command line, build tool, or API.
- Website: http://flywaydb.org/

Flyway Limitations

- Flyway cannot reset the database to a known state.
- Flyway expects you to order versions through file name conventions. This is unwieldy with large numbers of scripts in each migration. Inserting new scripts in correct location is painful.
- Flyway does not track environments.
- Flyway has no notion of deployment phases.
- Code must be present in each script. Over time this can lead to lots of code duplication and inferior use of source code control.
- Flyway cannot build a single deployment file that DBAs can use.
- How to solve these problems: Use gradle, groovy, and git to enhance and extend the core flyway API.

Resetting The Database

- Figure out how this is done for each database.
- Script doing this with groovy.
- Make script repeatable by making it a gradle plugin.
- Invoke this process using a gradle task.
- Example: db-reset project source code, usage in flyway-dsl-demo/build.gradle
- Future enhancements: Include Oracle resets, accept contributions for other databases.

Versioning

- Problem #1: If you have files with versions 1-20 for a migration and you want to insert a file at position 8, how many renames will you have to do?
- Problem #2: Different branches in source code control can lead to problems if you don't always have unique file versions.
- Solutions:
  - Build migration versions automatically based on position inside dsl file.
  - Base version number on project version, execution phase, and position inside execution phase.
  - Provide tools for bumping versions in an orderly way.
- See VersionBumper.groovy and MigrationStage.groovy in flyway-dsl; build.gradle in flyway-dsl-demo. Every migrate.groovy file will have ordering information automatically.

Environments and Phases

- DSL allows for environments and phases to be decided by user of dsl; use whatever makes sense for your organization.
- Migration tool will only run the phases that you tell it to run.
- Migration tool will only run the phases appropriate to your environment.
- DBA's can define one property file per environment.
- See migrate.groovy; Dsl.groovy and HistoryArea.groovy in flyway-dsl.

Source Code Control

- Every script must be included a migration for it to be run.
- But, it's best to keep a single version of Oracle packages, SQL Server procedures under source control.
- Solution: Do the right thing and keep files under source code control, tell the DSL to include them in a migration when necessary. This is why the “resource:” named parameter exists in flyway-dsl.
- Solution: Keep sql files under a separate **sql** directory, just like your would keep java files in a **java** directory or groovy files in a **groovy** directory.
- Solution: Use a distributed version control system just like you would for any other source code. I like git, but mercurial would also work fine.
- See: any migrate.groovy file that uses the “resource:” parameter.

Single Versioned Distributable

- Build tools should not be installed in a production environment. Your production control teams should not be executing maven or gradle build files.
- DBA's prefer traditional tools they are familiar with and usually reserve the right to inspect code before scripts are run.
- Solution: Build a simple zip file that contains shell scripts they can run from anywhere plus all sql files in a human readable format.
- Solution: Extend gradle's application plugin to build a zip file containing all necessary jars, sql scripts, plus a simple shell script or bat file to wire it all together.
- See: FlywayDslPlugin, especially the enableDistZip method.

Take Home Message

- Don't accept manual and error prone deployments, do something about it.
- I think this is a great way to migrate databases and is essential if you want to do continuous delivery.
- Gradle and groovy are incredibly flexible and enable some amazing stuff, they almost make DevOps fun.
- DevOps is kind of a dirty word, but 1) It's really only something developers can do well and 2) Is only going to get more important as applications become even more distributed, virtualized, and “cloudy”.
- Code is moving closer to the data more and more often. Traditional app server deployments don't make sense in these environments and will necessitate something **like** flyway-dsl if you want maintain your sanity.
- Even if flyway-dsl isn't useful to you, the concepts of small DSL's and gradle plugins can probably make your life easier.

Facts of Software Development

Developers spend a lot of time designing code, writing code, testing code, refactoring code, integrating code, and deploying code. To the extent that we care about software development (and if you are viewing this presentation you **do**), then it means you care about your code.

Developers tend to not spend much time thinking about databases. We prefer databases that are 1) hidden behind ORM, 2) managed by other people, and 3) devoid of business logic.

To the extent that developers think about deploying applications, this is almost always thinking about **code** deployment and usually not thinking about **database** deployment.

Ironies of Software Development

Companies (i.e. things that write paychecks) care a lot more about databases than about applications. They will tolerate bad UI's and inconvenient applications, but they will not tolerate data loss.

The databases inside of a company are going to live a lot longer than the applications that use them. The data inside the databases are to a near approximation expected to have an infinite lifespan, unless laws require them to be purged.

Ultimate Irony: The thing companies care about most are what developer's think about least. There is a lot of power and influence in influencing how data is used, deployed, migrated, etc. Developers are in a great position to wield this power. My experience is that DBA's will cede a lot of power if you can make their life easier and they trust you. Hence, flyway-dsl.

Images Used

Flyway Logo: http://flywaydb.org/
Gradle Logo: http://en.wikipedia.org/wiki/Gradle
Groovy Logo: http://en.wikipedia.org/wiki/Groovy_(programming_language)
Git Logo: http://en.wikipedia.org/wiki/Git_(software)
Devil from Codex Gigas: http://en.wikipedia.org/wiki/Devil
Coninuous Delivery book cover:
http://www.amazon.com/dp/0321601912?tag=contindelive-20
Continuous Delivery:
http://continuousdelivery.com/wp-content/uploads/2014/02/04_CD_managing_data_low-res.jpg

Me: David Clark

My Stuff: https://github.com/dwclark

Code Repos: db-reset, flyway-dsl, flyway-dsl-demo

Presentation In: flyway-dsl-demo