DBdeployer

About the Speaker:

CJ Estel, Operations Database Engineer
CoverMyMeds
cj@cjestel.net

Github:

https://github.com/covermymeds/dbdeployer

Company Tech Blog:

https://www.scriptscribe.org/

Some Common Problems

- Lots of environments (testX, alpha, beta, UAT, staging, production, sandbox, etc) need updated with the same set of changes, but have different SLA's and deployment schedules
- DBA's need early visibility into changes to audit them for normalization, architecture, and impact
- DBA's need to review schema changes before they get deployed to reduce production impact

Problems Continued

- Application users should not have more permissions than they require by the application, therefore they probably shouldn't be the user adding/dropping tables or fields
- Developers would like to be able to stand up a database that has minimal seed data allowing them to work and test locally
- Different environments could have different seed data, different schemas or different table structures

Problems Continued

- Some replication strategies do not easily support DDL and require a coordinated deployment effort
- Some companies have different database server types and this typically involves different deployment strategies
- Deployments must meet company change control requirements

Tracking Production Changes

- Changes shouldn't be able to be made to the database outside of:
  - the application
  - an administrative interface
  - a peer reviewed change deployed through a known and tested process
  - directly through change control

Issues with Ruby Migrations

- Intentionally built to use as generic of sql as possible which may not take advantage of features available to your system
- Has no regard for how long tables may be locked during a migration against a large table
- Can reduce visibility on what sql will actually be deployed to servers
- Doesn't address or track manual changes to data caused by application bugs
- Migrations can be difficult to stage when not using a monolithic database
- If more than one app uses the same database, which app manages changes

What is DBdeployer

- A command line utility written in-house that:
  - Executes sql via the native database binary and is compatible with MSSQL, Postgres, MySQL, and is extensible to other RDBMS's
  - Tracks deployments to each database
  - Meets change control requirements
  - Has a one touch button to bring a database to current state
  - Allows for one off deployments to fix application data
  - Ensures we always have a "tiny db" for each database managed with it
  - Supports differences in a database based on server or environment

Advantages of DBdeployer

- Changes are reviewed as the sql that will be executed. This can help decide if an ORM generated the best sql to apply in prod
- Changes can be audited in advance so that ETL jobs or replication that needs staged is less likely to be affected
- Database changes are scripted and can be deployed like a regular application
- Database changes are abstracted to the database layer instead of being managed by application

Advantages of DBdeployer (cont)

- Can be used to facilitate staged rollouts of database changes where trigger based replication is utilized (update idle/reporting node first)
- Can tie into the monitoring system to alert if an environment is out of date
- Environment specific data can be handled with DBdeployer

How Does DBdeployer Work?

- There are folder locations you can store sql files:
  - schema
  - seed
  - changes
  - rollback
  - archive

How Does DBdeployer Work (continued)?

- Written in bash which ensures it calls the database binaries the same way you would without relying on a driver layer or interpreter
- Tracks deployments in its own database
- Deployment ordering is based on the default alpha-numeric ordering of the file system. It is recommended to use a date-based naming convention for files

How Does DBdeployer Work (continued)?

- A quick demo...