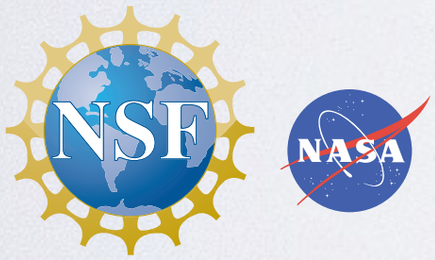
UNAVCO

MANAGING YOUR SCHEMA

Using migrations for consistency, repeatability, and sanity



Jeremy Smith

UNAVCO

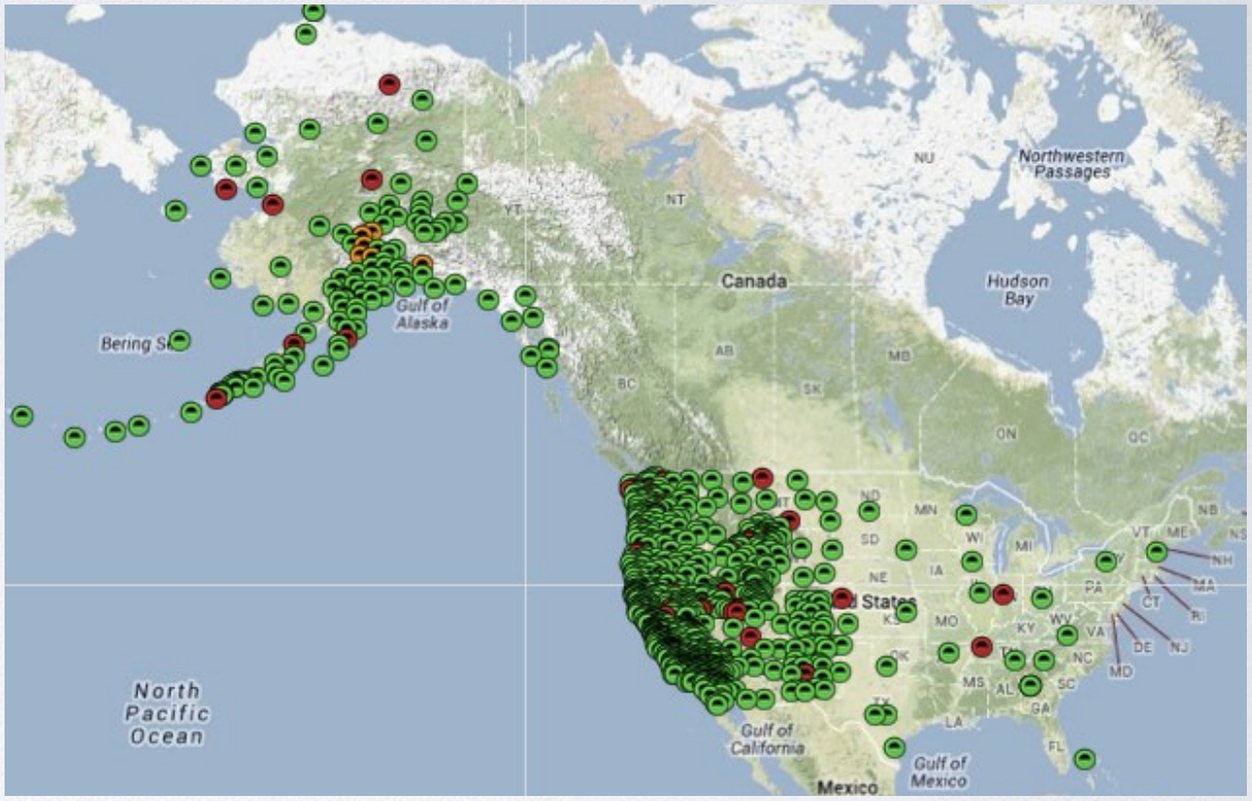
ABOUT ME

- Senior Software Engineer, Technical Lead
- ~Equal parts developer and DBA
- Background in geoscience
- Used migrations for ~7 years
- Contact: jsmith@unavco.org

ABOUT UNAVCO

- NSF and NASA funded
- Non-profit
- Consortium
- Facilitates geoscience research and education

www.unavco.org



ABOUT UNAVCO

- Used PostgreSQL for ~10 years
- Store detailed info about the stations, time series data, etc.
- ~500 tables, ~500 GB

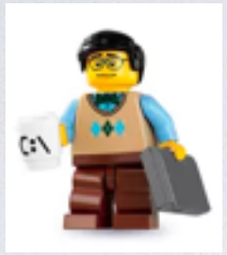
www.unavco.org

UNAVCO

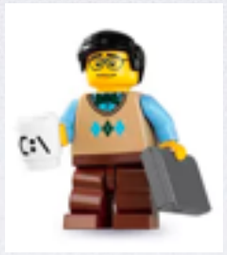
OUTLINE

- Why Use Migrations?
- Intro to Flyway
- Flyway usage
- Writing effective migrations
- Integration with Jenkins

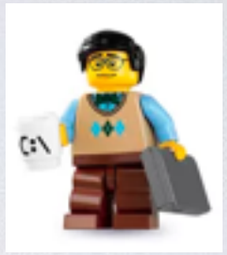
THE PROBLEM



Development Database



Development Database



Development Database

Continuous Integration Database(s)

↓

Test Database

↓

Production Database

Hotfixes

Image Source: http://www.amazon.com/

UNAVCO

THE PROBLEM

How To:

- Keep the databases in sync with each other?
- Synchronize software releases with database changes?
- Determine the current state of each database?
- Create a new development or test database?

UNAVCO

MIGRATIONS

- Migrations are versioned changes to your schema or data

V.1: CREATE TABLE USERS…

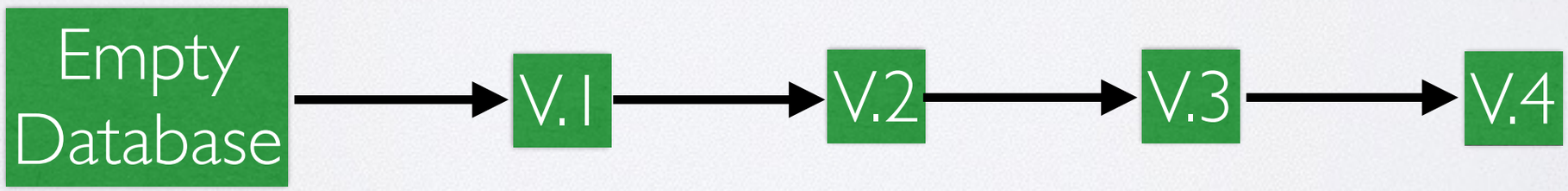
V.2: ALTER TABLE USERS…

V.3: CREATE INDEX ON TABLE USERS…

V.4: CREATE TABLE USER_TYPES…

MIGRATIONS

- Migration Versioning gives you:
- The current state of a database
- A clear upgrade path



UNAVCO

MIGRATIONS

- Migrations are code, should be kept in version control
- Version control gives you:
  - Ability to share changes with other developers/admins
  - Integration with build and test tools
  - History

UNAVCO

FLYWAY

http://flywaydb.org

- Migration Tool
- Open Source, ~5 years old
- Cross-platform
- Written in Java, but does not require Java knowledge
- Multiple interfaces: command line, Java API, Gradle, Ant, Maven
- Supports **PostgreSQL**, Oracle, SQL Server, MySQL, SQLite, etc.
- Does NOT support reversible migrations

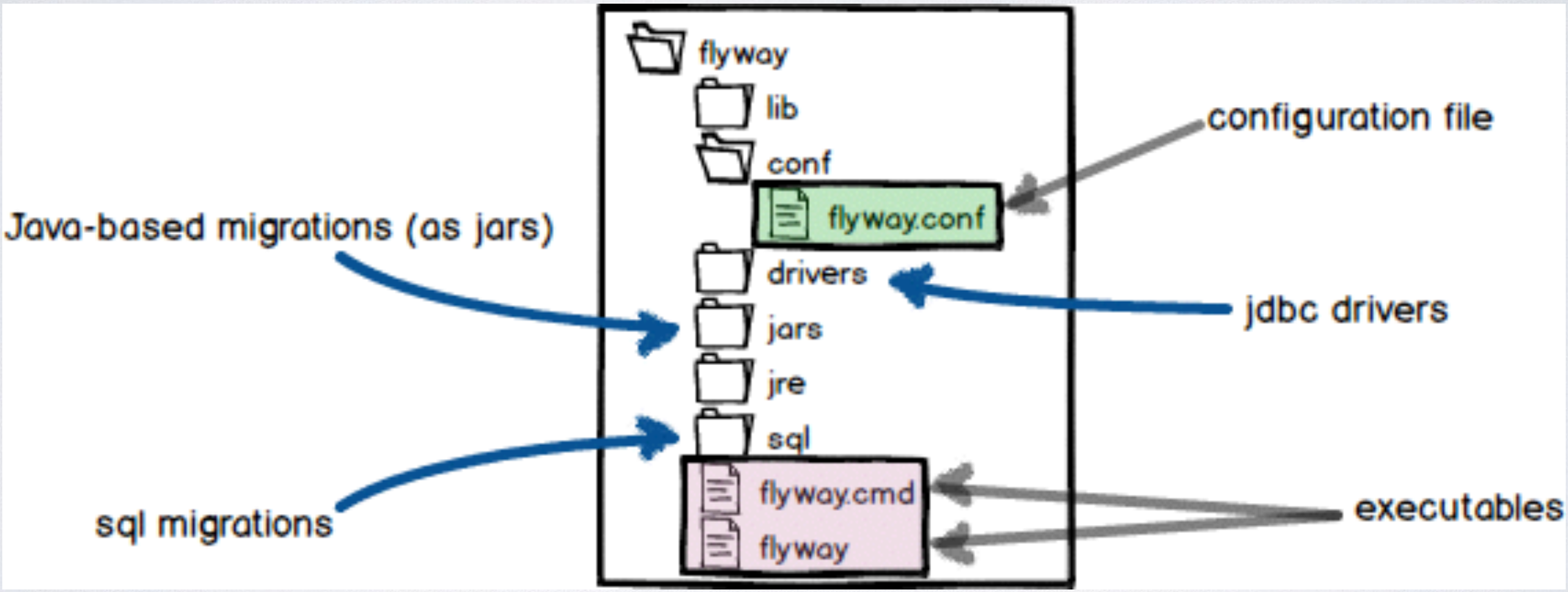
UNAVCO

FLYWAY COMMAND LINE

- For users who do not use the JVM or do not want to use a build tool (Gradle, Maven, etc.)
- Optionally bundled with the JRE for OS X, Windows, Linux
- Can also use system Java - best option for multiple operating systems
- Download from here: http://flywaydb.org/documentation/commandline/

FLYWAY COMMAND LINE

Directory Structure



Source: http://flywaydb.org/documentation/commandline/

UNAVCO

FLYWAY CONFIGURATION

- Highly configurable, but has good defaults
- Must set connection parameters
- All settings in both config file and command line parameters
- command-line parameters override items in config file

UNAVCO

FLYWAY: CREATING MIGRATIONS

- One file per migration
- Migrations can be written in plain SQL or Java
- Use Java migrations for advanced data changes
- Use SQL migrations for everything else
- No special languages or XML!

UNAVCO

HOW TO SET MIGRATION VERSION

Versions are determined from the filename

Example File Name:
V0002_00__Fix_things_I_broke_in_V1.sql

- **prefix**: default V
- **version**: version parts separated by dots or underscores
- **separator**: __ separates version and description
- **description**: describes the migration
- **suffix**: default sql

VERSIONING RECOMMENDATIONS

V0001_00__The_description.sql

- Leading zeros for lexical ordering

- Version parts let you break up large migrations

UNAVCO

HOW TO APPLY MIGRATIONS

flyway migrate

- Searches in specific folders for migrations
- Applies new migrations in order
- Each migration runs in its own transaction
- On failure, migration is rolled back and no further migrations applied

FLYWAY METADATA TABLE

First migrate command creates schema_version table

```
                    Table "public.schema_version"
     Column      |            Type             |       Modifiers
-----------------+-----------------------------+------------------------
 version_rank    | integer                     | not null
 installed_rank | integer                     | not null
 version         | character varying(50)       | not null
 description     | character varying(200)      | not null
 type            | character varying(20)       | not null
 script          | character varying(1000)     | not null
 checksum        | integer                     |
 installed_by    | character varying(100)      | not null
 installed_on    | timestamp without time zone | not null default now()
 execution_time | integer                     | not null
 success         | boolean                     | not null
Indexes:
    "schema_version_pk" PRIMARY KEY, btree (version)
    "schema_version_ir_idx" btree (installed_rank)
    "schema_version_s_idx" btree (success)
    "schema_version_vr_idx" btree (version_rank)
```

MIGRATE IMPORTANT OPTIONS

- **-url**: jdbc url to the database (e.g. jdbc:postgresql://localhost:5432/mydb)
- **-target**: maximum version (e.g. -target=0001.05, default is latest version)
- **-validateOnMigrate**: if true, verify checksum of all applied migrations (default: true)

UNAVCO

HOW TO GET CURRENT VERSION

flyway info

- Prints a table of migration status

- Shows applied (State=Success), new (State=Pending), and failed (State=Failure) migrations.

- Failure means the migration caused an error and could not be rolled back. Will not happen in PostgreSQL.

- Use as a test for the -target option before running migrate

UNAVCO

HOW TO START OVER

flyway clean

- Drops all objects in specified schemas

- **Never** run in production…

- **-schemas** option: comma separated list of schemas managed by flyway (default: default schema of connection)

- Previously missed some types of objects, but has improved significantly (does not handle extensions, see **https://github.com/flyway/flyway/issues/100#issuecomment-93705648** for a workaround)

UNAVCO

HOW TO CHECK APPLIED MIGRATIONS

flyway validate

- Check that applied migrations have not changed
- Checks names, types (sql or java), and checksums

flyway repair

- Repairs the flyway metadata table
- Corrects wrong checksums
- Removes failed migrations from metadata table

UNAVCO

HOW TO INTEGRATE FLYWAY WITH AN EXISTING DATABASE

It will take some work, but it's worth it

- Create a base version from the existing production database
- Dump out the schema (pg_dump —schema-only)
- Include reference data (pg_dump —data-only —table=…)
- Put schema and reference data into a base migration: V0001_00__Base_version.sql

More Info: http://flywaydb.org/documentation/existing.html

UNAVCO

HOW TO INTEGRATE FLYWAY WITH AN EXISTING DATABASE

- Run **flyway baseline** on production database
  - Adds schema_version table to existing database
    - Adds metadata to schema_version so flyway knows to skip migrations
- Use baselineVersion and baselineDescription options
- Start writing migrations with V0002

FLYWAY COMMANDS

- Migrate
- Clean
- Info
- Validate
- Baseline
- Repair

WRITING GOOD MIGRATIONS

Source: wikipedia.org

What type of changes do you think should be in migrations?

UNAVCO

WRITING GOOD MIGRATIONS

What should be in a migration?

- **All structural changes:** creating or altering tables, functions, indexes, views, etc.

- Reference data

- Users…?

THE PROBLEM WITH USERS

- Users are cluster-wide

- flyway clean does **not** remove users

- How to keep users and permissions in sync?

UNAVCO

THE PROBLEM WITH USERS

If you have one database per cluster, remove users after clean,

using a callback (flyway/sql/afterClean.sql)

```
do $$
    declare rolename text;
    begin
        for rolename in select rolname FROM pg_roles
          WHERE rolname NOT ILIKE '%postgres%'
          loop
            execute 'DROP ROLE ' || rolename;
          end loop;
    end $$;
```

UNAVCO

WRITING GOOD MIGRATIONS

Other Tips

- Keep them small, for readability
  - Use minor version numbers to break up big migrations
  - e.g. V0002_00__User_tables.sql, V0002_01__User_views.sql
- Skip some major version numbers if your project will not go into production for a while

UNAVCO

Jenkins

- A widely used tool for:
  - Automating builds
  - Automating Continuous Integration Tests
- Builds projects on change in source code or on a schedule

UNAVCO

INTEGRATION WITH JENKINS

Setting up your Jenkins Project

- Watch flyway directory and software directories for changes
- Works best with a cluster per Jenkins project
- Create a simple, fail-fast project to just build the db, which triggers more complex downstream projects

UNAVCO

INTEGRATION WITH JENKINS

Setting up your Jenkins Project

- Projects should be parameterized:
  - DB_HOST, DB_PORT, DB_NAME
- For simple databases, use clean and migrate:

```
${WORKSPACE}/flyway/flyway -url=jdbc:postgresql://${DB_HOST}:
${DB_PORT}/${DB_NAME} clean migrate
```

INTEGRATION WITH JENKINS

- Alternatively, use postgres tools:

```
dropdb --if-exists -U postgres -h ${DB_HOST} -p ${DB_PORT}
  ${DB_NAME}

psql -U postgres -h ${DB_HOST} -p ${DB_PORT} -d template1 -f
  "${WORKSPACE}/reset_users.sql"

createdb -U postgres -h ${DB_HOST} -p ${DB_PORT}
  encoding=UTF-8 ${DB_NAME}

${WORKSPACE}/flyway/flyway -url=jdbc:postgresql://${DB_HOST}:
${DB_PORT}/${DB_NAME} migrate
```

INTEGRATION WITH JENKINS

- If you need to load test data at specific schema versions, use multiple calls to migrate, with the target option:

```
${WORKSPACE}/flyway/flyway -url=jdbc:postgresql://${DB_HOST}:
${DB_PORT}/${DB_NAME} -target=0005.00 migrate

# Load data here

${WORKSPACE}/flyway/flyway -url=jdbc:postgresql://${DB_HOST}:
${DB_PORT}/${DB_NAME} migrate
```

UNAVCO

SUMMARY

- Migrations:
  - keep your databases in sync
  - Show the current state of the database
  - Allow you to test database changes

UNAVCO

SUMMARY

- Flyway:
  - is a powerful and flexible migration tool
  - applies migrations (in the proper order), shows you the current status, and can clean out schemas for you
- Use flyway with Jenkins to run integration tests when any schema or software change is checked in

UNAVCO

FINAL REMINDER

All structural changes must be in a migration!