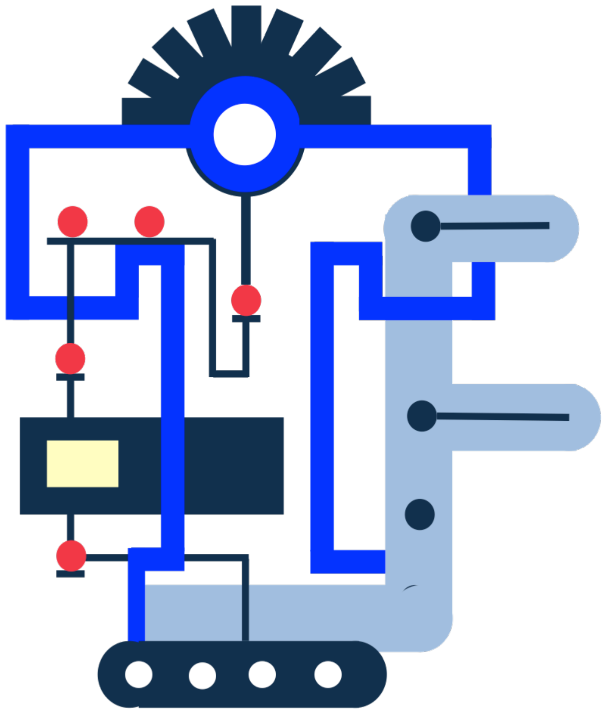
Twineworks



Automated testing of data integration solutions

Slawomir Chodnicki
BI-consultant

slawomir.chodnicki@twineworks.com

Day 1 overview

- hands-on project intro
- various kinds of automated tests
- testable solutions vs. untestable messes
- minimizing external dependencies
- fixtures & helpers
- declarative tests

Day 1 overview (continued)

Twine**works**

- Using ETL to test ETL solutions
  - PDI techniques for unit tests
  - PDI techniques for integration tests
  - practical limitations of ETL-based tests

Day 2 overview

Twineworks

- Scripting tests
  - jruby as a scripting language
  - scripting helpers for command execution and fixture loading

Day 2 overview (continued)

Twineworks

- Organizing the test suite with rspec
  - rspec output
  - rspec features
  - rspec reports

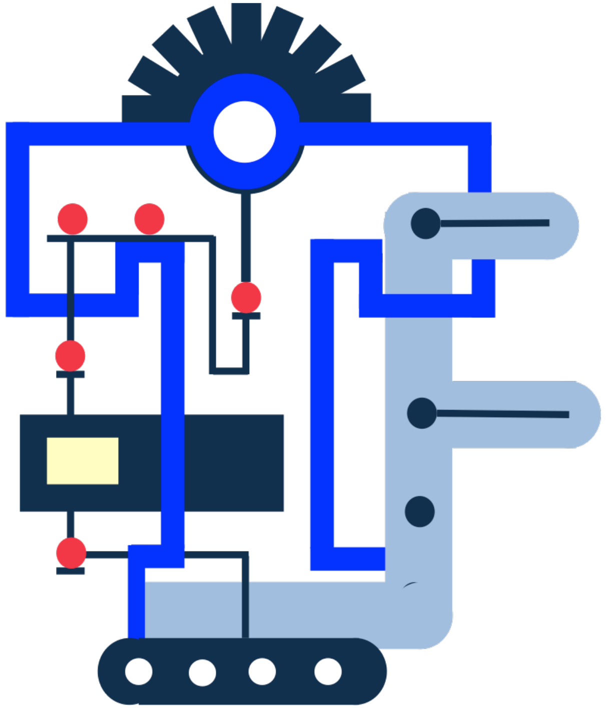
Day 3 overview

- Continuous Testing with Jenkins
  - installing jenkins
  - access control in jenkins
  - useful jenkins plugins
  - setting up jenkins to run ETL tests
  - test reports

Day 3 overview (continued)

- Deployment
  - deploying from version control
  - migration scripts

Twineworks



Demo project

Hands-on project intro

Twine**works**

```
.
|-- bin             # entry point scripts and dependencies
|-- data            # source data ingested by the ETL
|-- environments    # environment configuration
|-- etl             # ETL solution
    `-- spec        # ETL test suite
|-- logs            # logging location
`-- spec            # jruby tests and helpers
```

database configuration

Twineworks

- database: ‘dwh’
- user: ‘etl’
- password: ‘password’

database configuration

Twine**works**

```
$ mysql -u root

CREATE DATABASE dwh;

GRANT ALL PRIVILEGES ON dwh.* TO etl@'%' identified by 'password';
GRANT ALL PRIVILEGES ON dwh.* TO etl@'localhost' identified by 'password';

FLUSH PRIVILEGES;
```

project configuration

- environment configuration in
  - environments/local
    - environment
    - my.cnf
    - .kettle/shared.xml
    - .kettle/kettle.properties

environment

Twineworks

```
#!/bin/bash
export ROBOT_DB_NAME=dwh
export ROBOT_LOG_BASE_DIR=/Users/slawo/Desktop/etl-testing/logs
export ROBOT_PDI_HOME=/Users/slawo/pentaho/data-integration-5.4
```

my.cnf

```
[client]
user=etl
password=password
host=localhost
```

.kettle/kettle.properties

Twine**works**

```
### Settings for Robot
ENV_MACHINE=local
ENV_DATA_DIR=/Users/slawo/Desktop/etl-testing/solution/data/in

# setting hostname explicitly shortens PDI startup time
KETTLE_SYSTEM_HOSTNAME=localhost
```

.kettle/shared.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<sharedobjects>
    <connection>
        <name>dwh</name>
        <server>localhost</server>
        <type>MYSQL</type>
        <access>Native</access>
        <database>dwh</database>
        <port>3306</port>
        <username>etl</username>
        <password>Encrypted 2be98afc86aa7f2e4bb18bd63c99dbdde</password>
        …
    </connection>
</sharedobjects>
```

Initialize db

Twineworks

```
$ bin/robot db reset

clearing database [ OK ]
initializing database
2017/06/20 15:32:37 - Kitchen - Start of run.
2017/06/20 15:32:37 - reset_dwh - Start of job execution
…
…
2017/06/20 15:32:41 - Kitchen - Finished!
2017/06/20 15:32:41 - Kitchen - Processing ended after 4 seconds.
[ OK ]
```

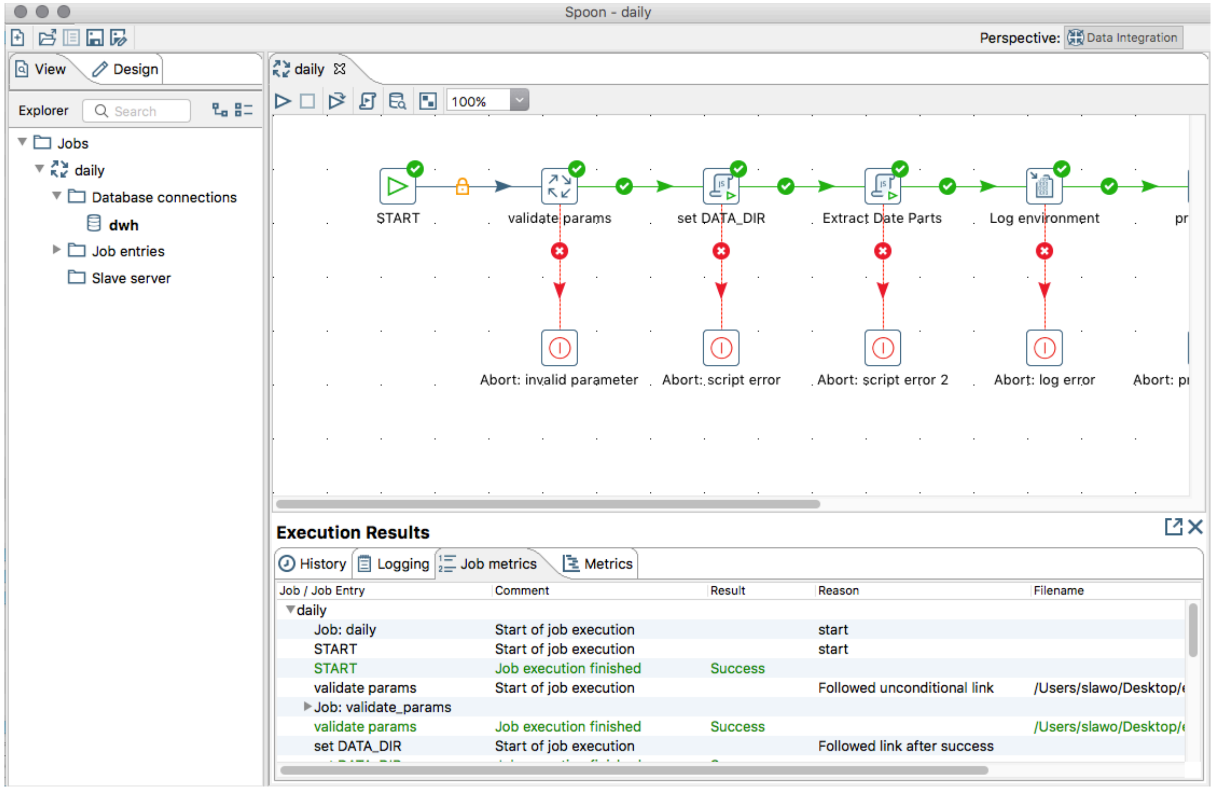
Run daily

Twineworks

```
$ bin/robot spoon
```

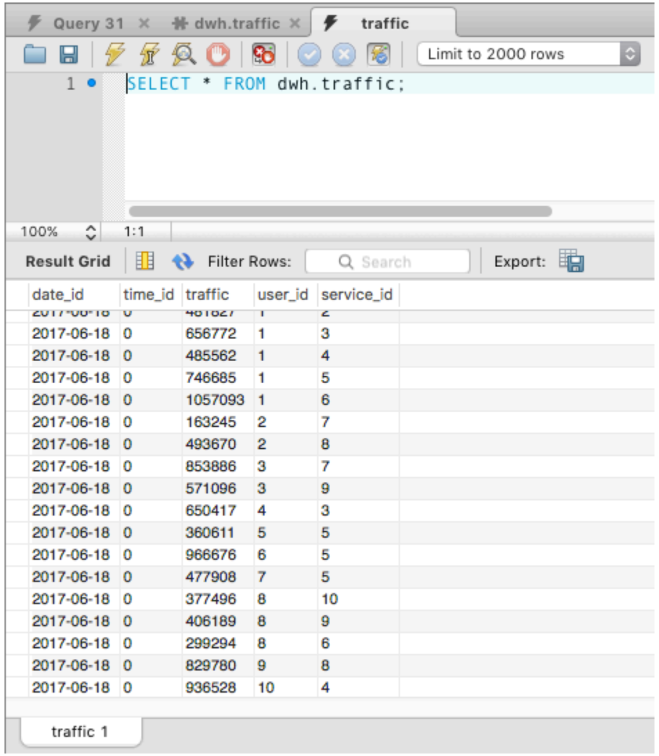
Run etl/daily.kjb

Twine**works**



Job / Job Entry	Comment	Result	Reason	Filename
daily				
Job: daily	Start of job execution		start	
START	Start of job execution		start	
START	Job execution finished	Success		
validate params	Start of job execution		Followed unconditional link	/Users/slawo/Desktop/
Job: validate_params				
validate params	Job execution finished	Success		/Users/slawo/Desktop/
set DATA_DIR	Start of job execution		Followed link after success	

Inspect database contents

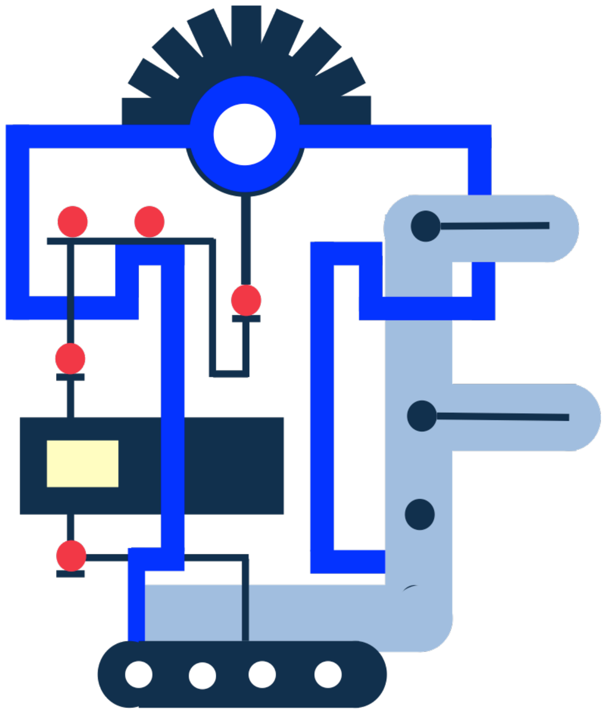

Query 31 | dwh.traffic | traffic

Limit to 2000 rows

```
SELECT * FROM dwh.traffic;
```

date_id	time_id	traffic	user_id	service_id
2017-06-18	0	656772	1	3
2017-06-18	0	485562	1	4
2017-06-18	0	746685	1	5
2017-06-18	0	1057093	1	6
2017-06-18	0	163245	2	7
2017-06-18	0	493670	2	8
2017-06-18	0	853886	3	7
2017-06-18	0	571096	3	9
2017-06-18	0	650417	4	3
2017-06-18	0	360611	5	5
2017-06-18	0	966676	6	5
2017-06-18	0	477908	7	5
2017-06-18	0	377496	8	10
2017-06-18	0	406189	8	9
2017-06-18	0	299294	8	6
2017-06-18	0	829780	9	8
2017-06-18	0	936528	10	4

Twineworks



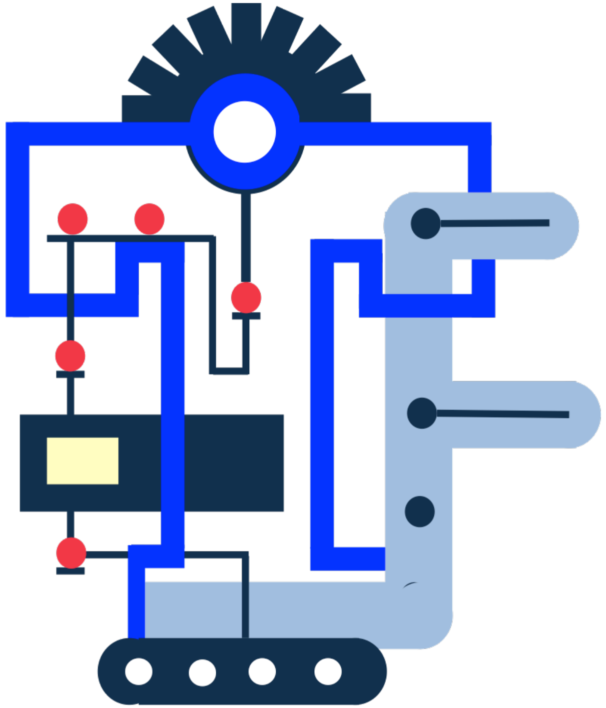
Testing ETL solutions

Kinds of automated tests

Twineworks

- Unit tests
- Integration tests
- Functional tests
- Non-functional tests

Twineworks

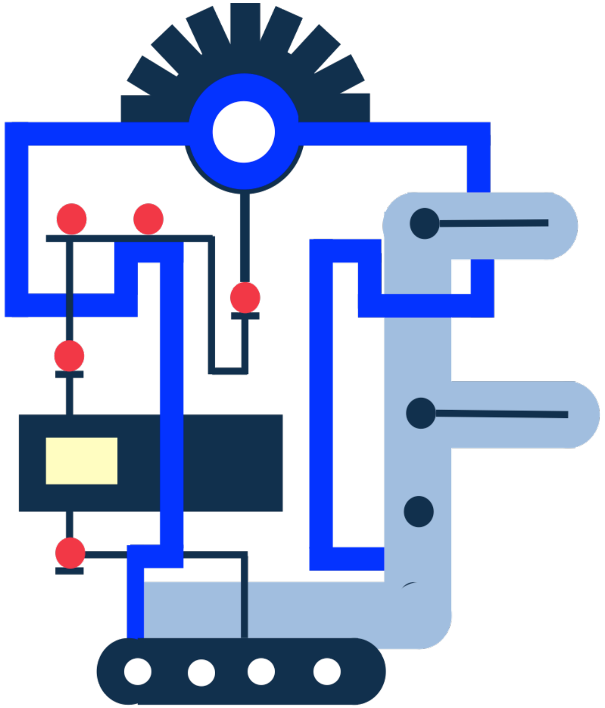


Unit tests

Unit tests

- Single isolated unit of ETL under test
- A unit performs a computation (no side-effects)
- What is a “unit” in PDI?
  - Job?
  - Transformation?
  - Sub-transformation (mapping)?

Twineworks



Testing jobs

A simple unit test

Create a job

`etl/spec/dwh/validate_params/validate_params_spec.kjb`

```
The job calls
```

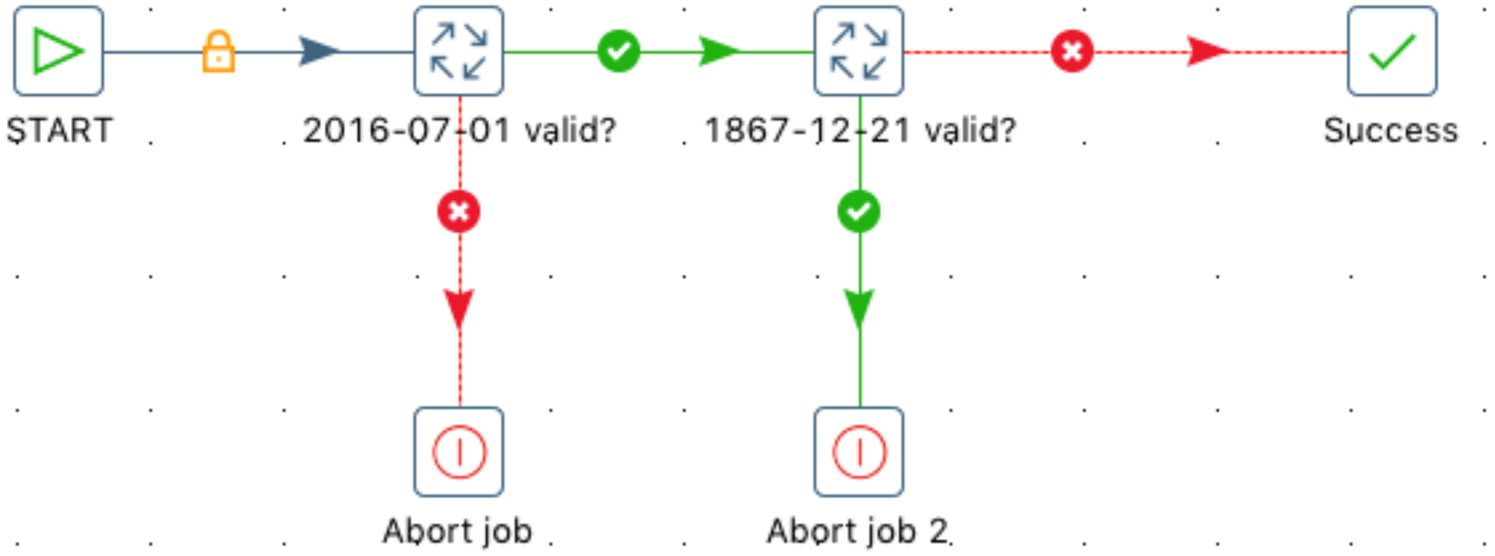
`etl/dwh/validate_params.kjb`

```
    - with DATA_DATE=2016-07-01 and expects it to succeed
    - with DATA_DATE=1867-12-21 and expects it to fail
```

The job succeeds if all expectations are met. It fails otherwise.

A simple unit test

Twineworks



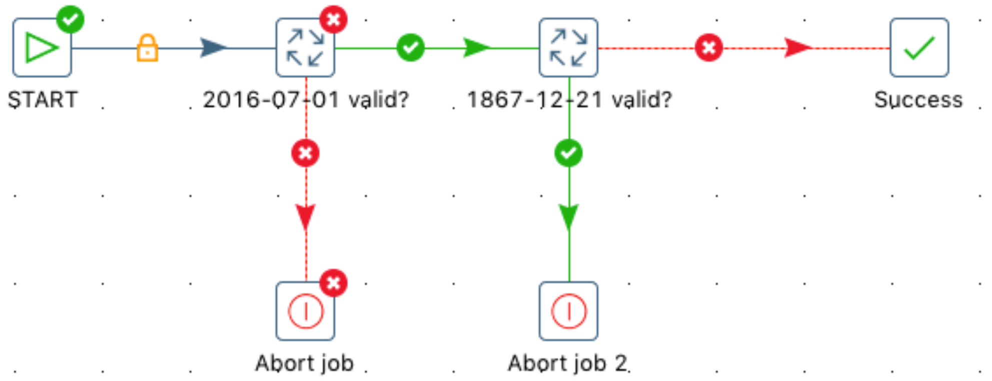
Test must run in known env

Twineworks

$ bin/robot db clear

Run

`etl/spec/dwh/validate_params/validate_params_spec.kjb`



Environment reset

Twineworks

Create a job

```
etl/spec/support/reset_all.kjb
```

The job runs

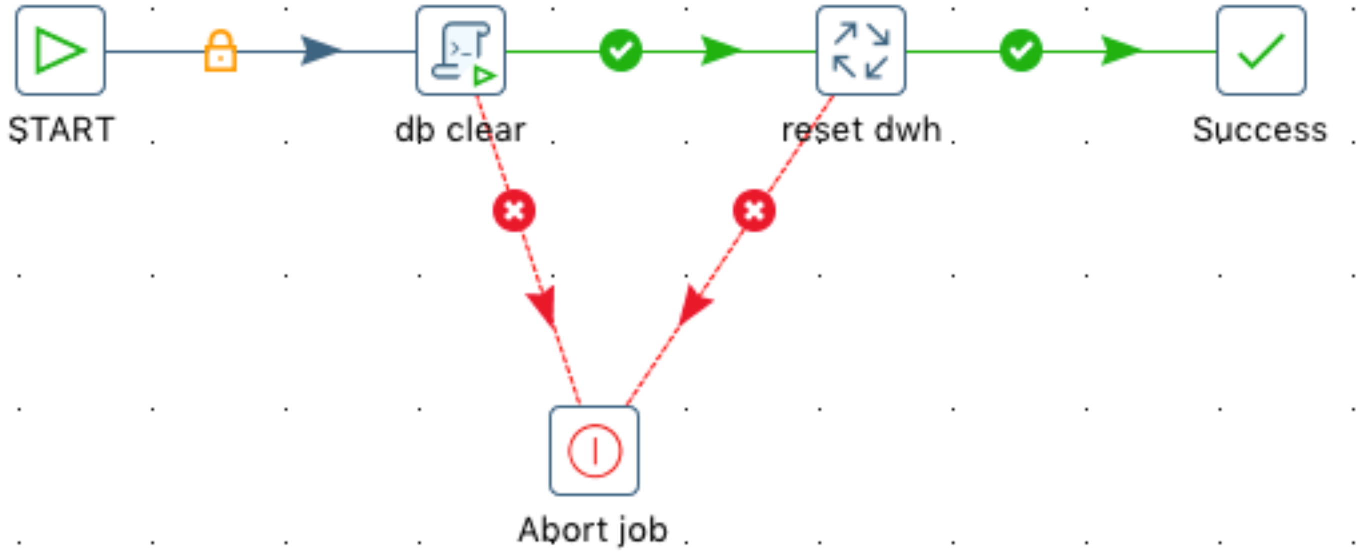
```
$ bin/robot db clear
```

Then calls

```
etl/init/dwh/reset_dwh.kjb
```

The job succeeds if both tasks succeed. It fails otherwise.

Environment reset



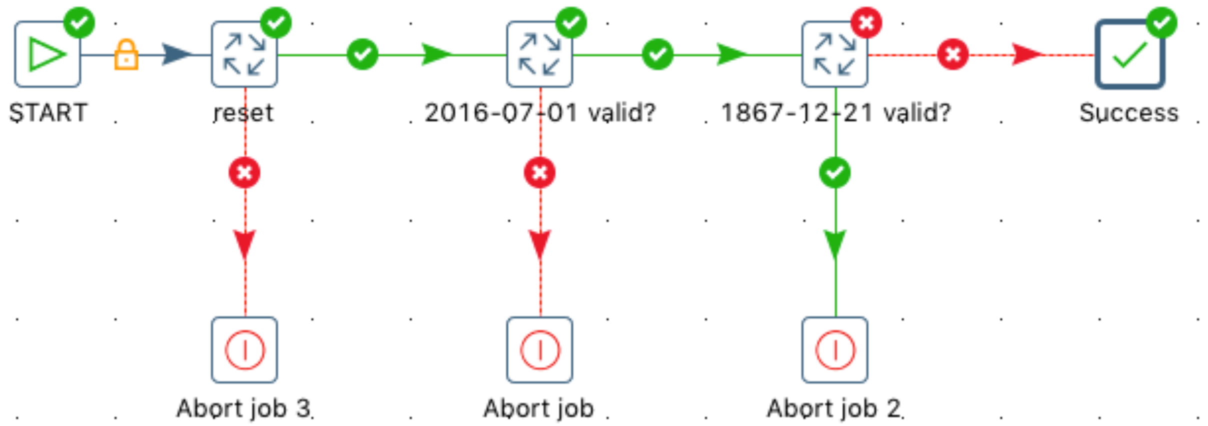
Test must run in known env

Twineworks

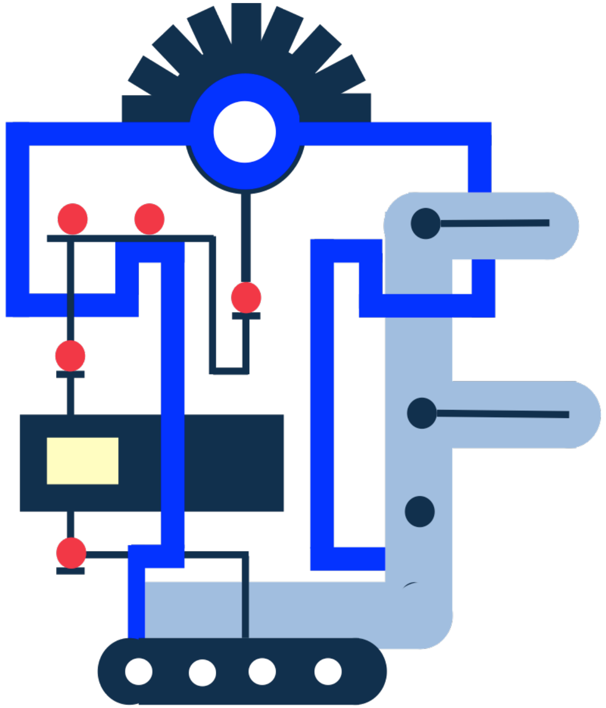
Fix

etl/spec/dwh/validate_params/validate_params_spec.kjb

to establish a known environment before testing.



Twineworks

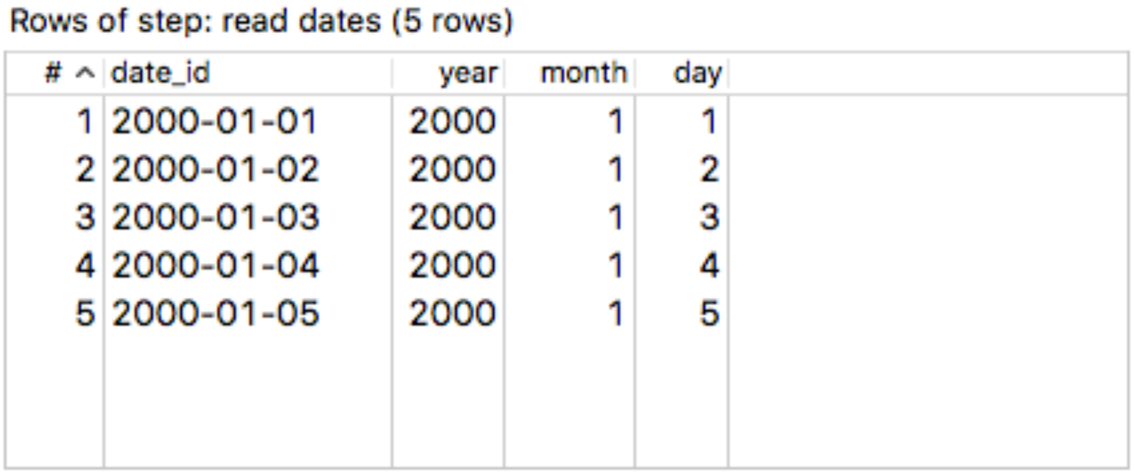


Testing transformation results

Test transformation results

Have a look at

`etl/util/day_sequence.ktr`



Rows of step: read dates (5 rows)

# ^	date_id	year	month	day
1	2000-01-01	2000	1	1
2	2000-01-02	2000	1	2
3	2000-01-03	2000	1	3
4	2000-01-04	2000	1	4
5	2000-01-05	2000	1	5

Test transformation results

Create a job
`etl/spec/util/day_sequence/day_sequence_spec.kjb`

The job calls
`etl/util/day_sequence.ktr`
With parameters

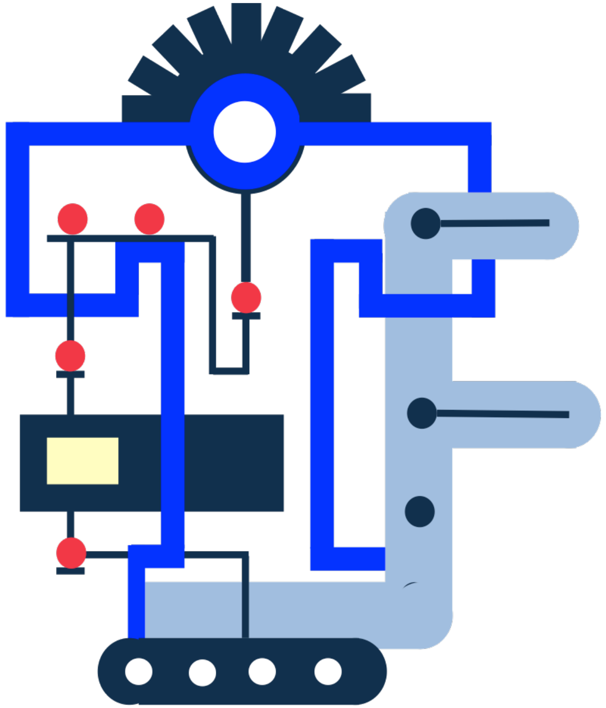
```
START_DATE=2017-02-27
DAY_COUNT=3
```

The job succeeds if and only if the transformation returns:

- exactly 3 rows, representing 2017-02-27, 2017-02-28, 2017-03-01
- with at least 4 fields each
  - string date_id
  - int year
  - int month
  - int day

Hint: the JavaScript job entry exposes result rows as ‘rows’.
http://wiki.pentaho.com/display/EAI/JavaScript+%28job+entry%29

Twineworks



Testing sub-transformations

Test sub-transformations

Twine**works**

Have a look at

`etl/util/string_cleaner.ktr`

It transforms a string to be a valid identifier in an internal system. The system accepts identifiers of the following form:

Only ascii letters a-z, A-Z, digits 0-9 and the underscore _ are allowed. The identifier may not start with a digit.

Test sub-transformations

Twineworks

Conversion is specified as follows:

Diacritics are reduced to the base character. So Ä becomes A, é becomes e, etc.
Strings beginning with a digit are prefixed with an underscore character.

The following characters are replaced by letter sequences:

&	_and_	$	_dollar_
*	_star_	£	_pound_
-	_dash_	€	_euro_
%	_percent_	.	_dot_
@	_at_	,	_comma_
#	_hashtag_		

Any other invalid character is replaced with an underscore character.

Test transformation results

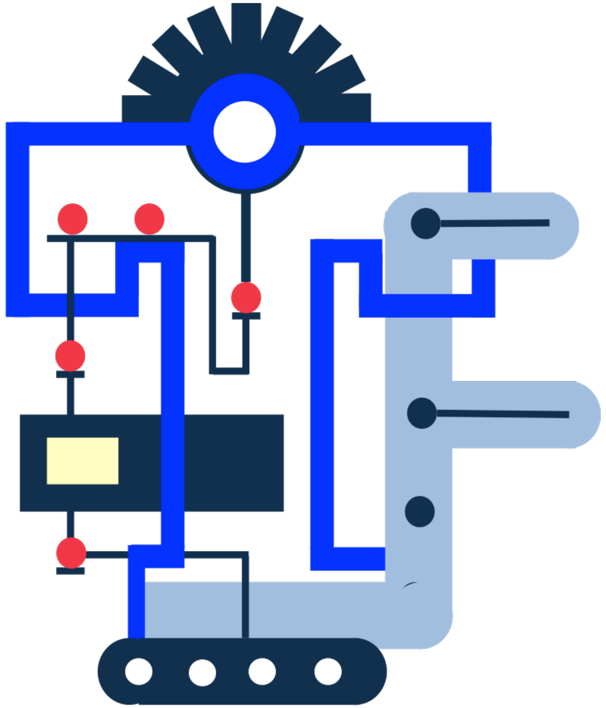
Create a job

`/etl/spec/util/string_cleaner/string_cleaner_spec.kjb`

The job verifies that **`etl/util/string_cleaner.ktr`** performs the conversion as per the input and expected columns of **`/etl/spec/util/string_cleaner/string_cleaner_spec.csv`**

The csv file is UTF-8 encoded.

Twineworks



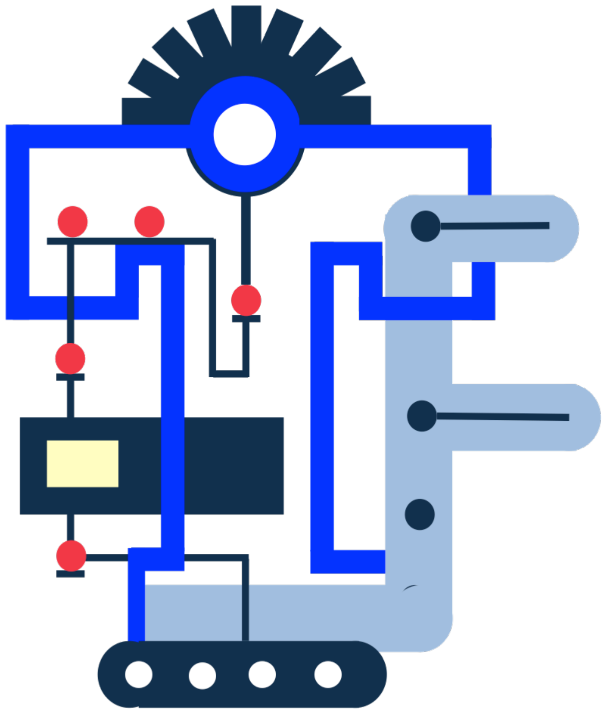
Integration tests

Integration tests

Twine**works**

- ETL responsible for a set of related side-effects under test
- Most common case in ETL testing
  - Test individual phases of a batch process

Twineworks



Tests for the pre-flight phase

Test the pre-flight phase

Twineworks

Have a look at

`/etl/dwh/pre_flight/pre_flight.kjb`

Given a DATA_DIR, DATA_DATE, as well as the corresponding DATA_YEAR, DATA_MONTH, and DATA_DAY. It verifies that a non-empty file exists at location:

`DATA_DIR/DATA_YEAR/DATA_MONTH/DATA_YEAR-DATA_MONTH-DATA_DAY.csv`

Test transformation results

Twine**works**

Create a job

`etl/spec/dwh/pre_flight/pre_flight_ok_spec.kjb`

The job verifies that **`/etl/dwh/pre_flight/pre_flight.kjb`** succeeds when the file specified by the paramaters exists.

Use the folder /**`spec/fixtures/pre_flight`** to act as DATA_DIR to host any test files.

Test transformation results

Twine**works**

Create a job

`etl/spec/dwh/pre_flight/pre_flight_empty_file_spec.kjb`

The job verifies that **`/etl/dwh/pre_flight/pre_flight.kjb`** fails when the file specified by the paramaters exists, but is empty.

Use the folder /**`spec/fixtures/pre_flight`** to act as DATA_DIR to host any test files.

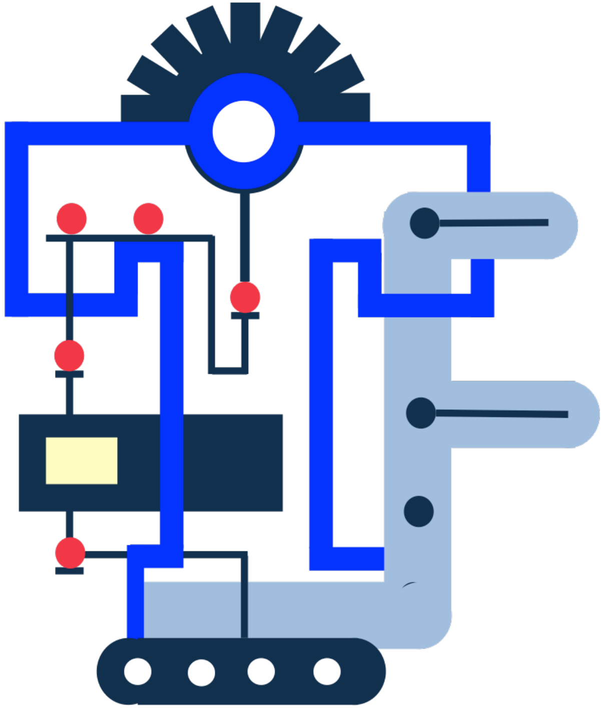
Test transformation results

Create a job

`etl/spec/dwh/pre_flight/pre_flight_missing_file_spec.kjb`

The job verifies that **`/etl/dwh/pre_flight/pre_flight.kjb`** fails when the file specified by the paramaters does not exist.

Twineworks



Tests for the stage phase

Test the stage phase

Have a look at

`/etl/dwh/stage/stage.kjb`

Given a DATA_DIR, DATA_DATE, as well as the corresponding DATA_YEAR, DATA_MONTH, and DATA_DAY. It loads the contents of the file **`DATA_DIR/DATA_YEAR/DATA_MONTH/DATA_YEAR-DATA_MONTH-DATA_DAY.csv`** into the database table: **stage_traffic**.

Test the stage phase

Twineworks

Create a job

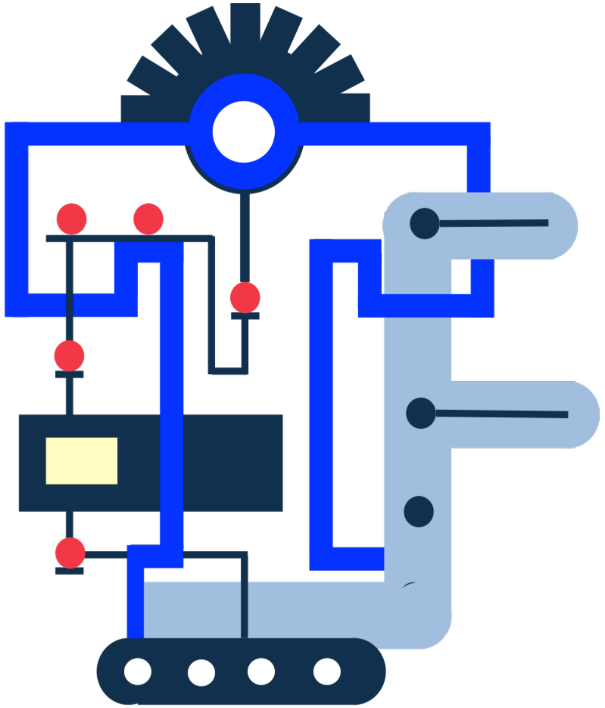
`etl/spec/dwh/stage/stage_spec.kjb`

The job verifies that **`/etl/dwh/stage/stage.kjb`**

- succeeds when a valid file is loaded
- table **stage_traffic** contains the expected number of records
- table **stage_traffic** contains one specific record from the loaded file

Use the folder /**`spec/fixtures/stage`** to act as DATA_DIR to host any test files.

Twineworks



A fixture helper

A fixture helper

Twineworks

Have a look at

`/fixtures/load/2017-06-21.json`

```
{
  "stage_traffic": [
    {
      "date_id": "2017-06-21",
      "time_id": 4,
      "user": "Gulliver",
      "service": "Libslack",
      "traffic": 100
    },
    {
      "date_id": "2017-06-21",
      "time_id": 4,
      "user": "Gulliver",
      "service": "Libslack",
      "traffic": 200
    },
    ...
  ]
}
```

A fixture helper

Twine**works**

```
{
  ”table_name_1": [
    {
      ”field_1": ”value_1",
      ”field_2": value_2,
      ”field_3": ”value_3",
      ”field_4": ”value_4",
      ”field_5": value_5
    },
    {
      ”field_1": ”value_1",
      ”field_2": value_2,
      ”field_3": ”value_3",
      ”field_4": ”value_4",
      ”field_5": value_5
    },
    ...
  ]
}
```

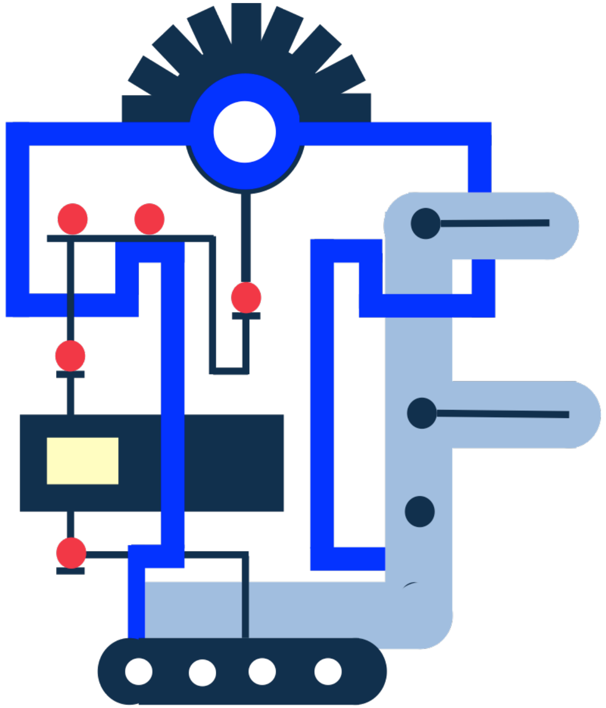
A fixture helper

Create a transformation

`etl/spec/support/load_json_fixture.ktr`

Given a JSON file of the above structure in parameter INPUT_FILE, it loads the data from the json file into the dwh database.

Twineworks



Test the load phase

Test the load phase

Twine**works**

Have a look at

`/etl/dwh/load/load.kjb`

Given a DATA_DATE, it deletes any records with that date_id from the traffic table, then aggregates records from **stage_traffic**, calculates the user_tag field, populates dimension tables **user**, and **service**, and writes fact records to fact table **traffic**.

Test the load phase

Twine**works**

Create a job

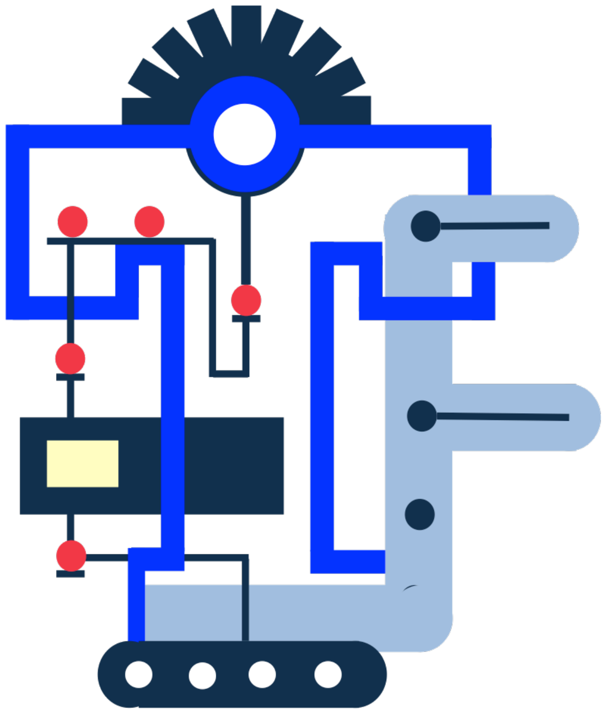
`etl/spec/dwh/load/load_spec.kjb`

The job verifies that **`/etl/dwh/load/load.kjb`**

- loads data from stage_traffic
- has aggregated the records for same date, time, user and service
- has calculated the user_tag by transforming it as expected
- has filled the service table as expected

Use **`/fixtures/load/2017-06-21.json`** to populate the **stage_traffic** table.

Twineworks

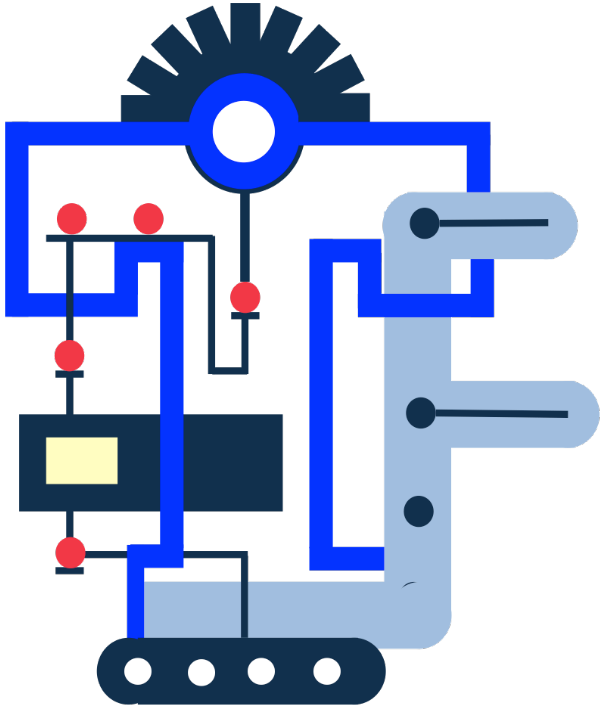


Functional testing

Functional tests

- Entry point of ETL solution under test
- Assertions reflect functional contract
  - Behavior on happy path
  - Behavior on errors
  - Behavior on incorrect invocation

Twineworks



A comparison helper

A comparison helper

Have a look at

`/spec/fixtures/functional/expected/2017-06-20.csv`

It contains the following columns

Name	Type
date_id	String
time_id	Integer
user	String
user_tag	String
service	String
traffic	Integer

This structure corresponds a full join of all data warehouse traffic information. Dimension keys are omitted.

A comparison helper

Create a transformation

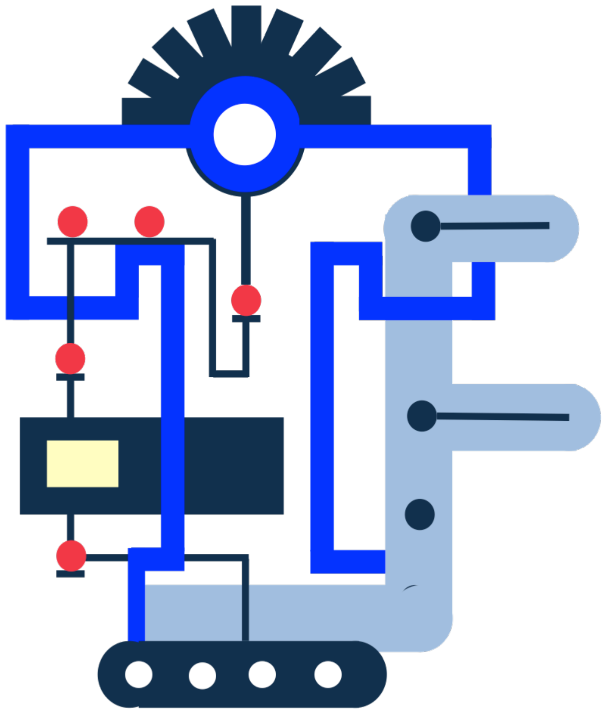
`etl/spec/support/compare_state_traffic.ktr`

Given a csv file of the above structure in parameter EXPECTED_FILE, it compares the data from the csv file with the contents of the dwh database.

The transformation succeeds when the contents are equal.

The transformation fails when the contents are different.

Twineworks



Test the daily run

Test the daily run

Twineworks

Have a look at

`/etl/daily.kjb`

Given a DATA_DATE as well as a DATA_DIR, it loads the contents of the corresponding file into the data warehouse.

DATA_DATE must exist in the date dimension table, and the corresponding file must exist in DATA_DIR.

Test the daily run

Twine**works**

Create a job

`etl/spec/daily/daily_0000_00_00_spec.kjb`

The job verifies that **`/etl/daily.kjb`** fails and does not load any data when invoked with invalid DATA_DATE 0000-00-00.

Test the daily run

Twine**works**

Create a job

`etl/spec/daily/daily_2001_01_01_spec.kjb`

The job verifies that **`/etl/daily.kjb`** fails and does not load any data when invoked with DATA_DATE 2001-01-01, for which the data file does not exist.

Test the daily run

Twine**works**

Create a job

`etl/spec/daily/daily_2017_06_20_spec.kjb`

The job verifies that **`/etl/daily.kjb`** succeeds and loads expected data when invoked with DATA_DATE 2017-06-20.

Use the folder **/spec/fixtures/functional** as DATA_DIR and **/spec/fixtures/functional/expected/2017-06-20.csv** to compare database state.

Test the daily run

Twine**works**

Create a job

`etl/spec/daily/daily_2017_06_20_reload_spec.kjb`

The job verifies that **`/etl/daily.kjb`** succeeds and loads expected data when invoked with DATA_DATE 2017-06-20 twice in a row. The expected data is the same as if the date had been loaded only once.

Use the folder **/spec/fixtures/functional** as DATA_DIR and **/spec/fixtures/functional/expected/2017-06-20.csv** to compare database state.

Test the daily run

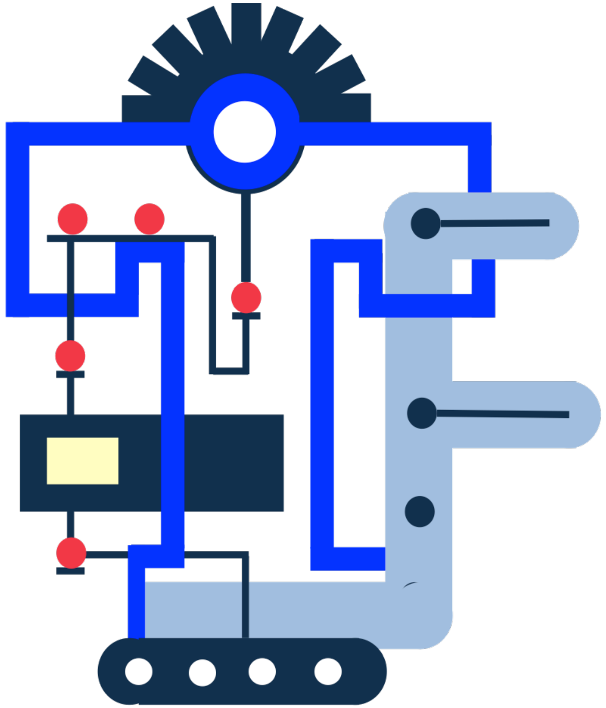
Create a job

`etl/spec/daily/daily_2017_06_20_to_21_spec.kjb`

The job verifies that **`/etl/daily.kjb`** succeeds and loads expected data when invoked with DATA_DATE 2017-06-20, and again with DATA_DATE 2017-06-21.

Use the folder **/spec/fixtures/functional** as DATA_DIR and **/spec/fixtures/functional/expected/2017-06-20-to-21.csv** to compare database state.

Twineworks



Non-functional tests

Non-functional tests

- Performance
  - how long does workload x take?
- Stability
  - what does it take to break it?
  - How much memory is too little?
  - What happens when loading unexpected data? (truncated file, column too long, 50MB XML in string field, badly formatted CSV reads as single field, empty files)

Non-functional tests

Twineworks

- Security
  - Verify configuration assumptions automatically
- Compliance
  - We must use version x of library y
  - Often done by inspection

Test configuration

Create a job

`etl/spec/environment/robot_dir_is_set_spec.kjb`

The job verifies that the kettle variable ROBOT_DIR is set correctly. The variable must point to an existing directory, and below that directory there must be a **bin/robot** file.

The job succeeds if above tests pass. It fails otherwise.

Test compliance

Twine**works**

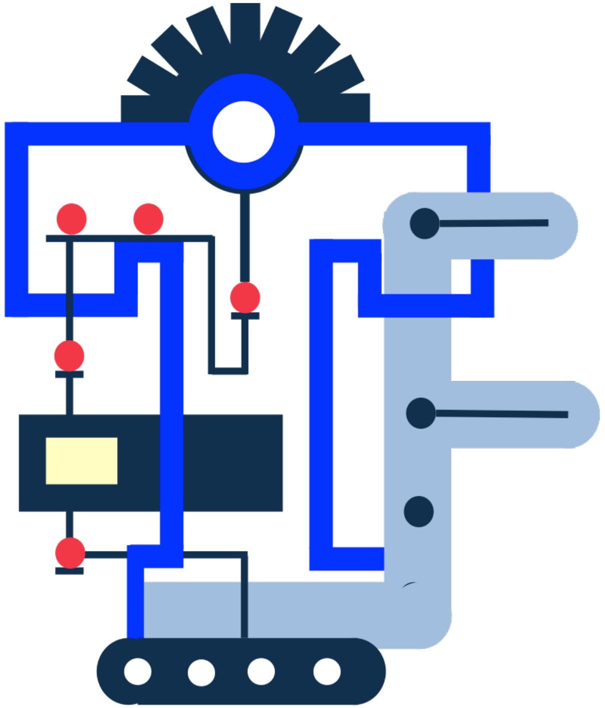
Create a job

`etl/spec/environment/mysql_driver_spec.kjb`

The job verifies that the file **${ROBOT_PDI_HOME}/lib/mysql-connector-java-5.1.42-bin.jar** exists.

The job succeeds if above test passes. It fails otherwise.

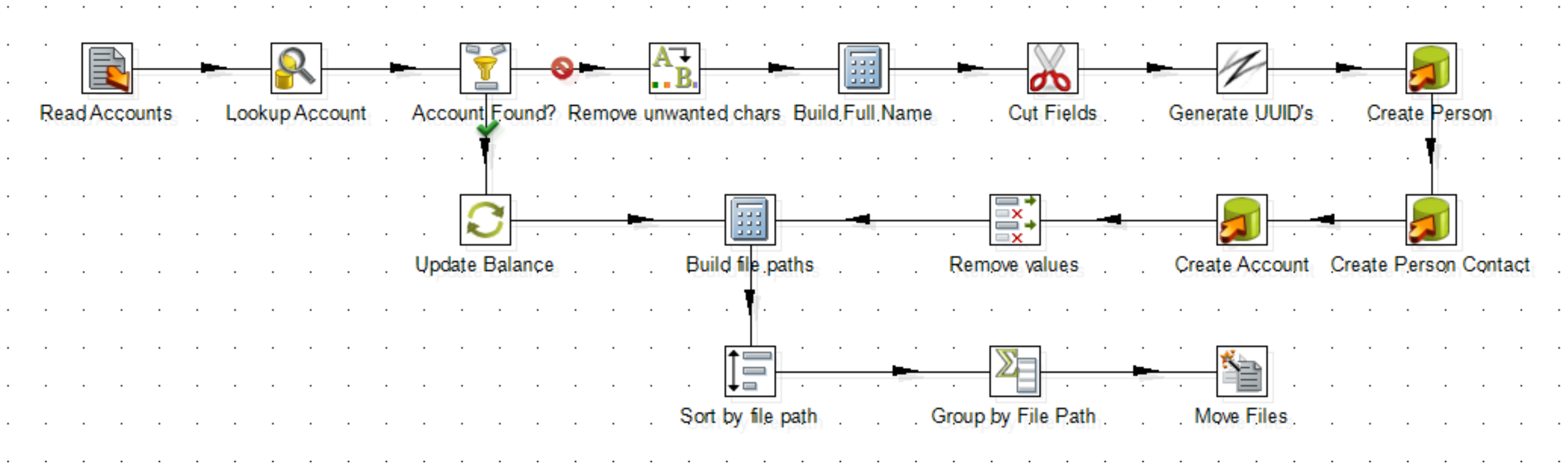
Twineworks



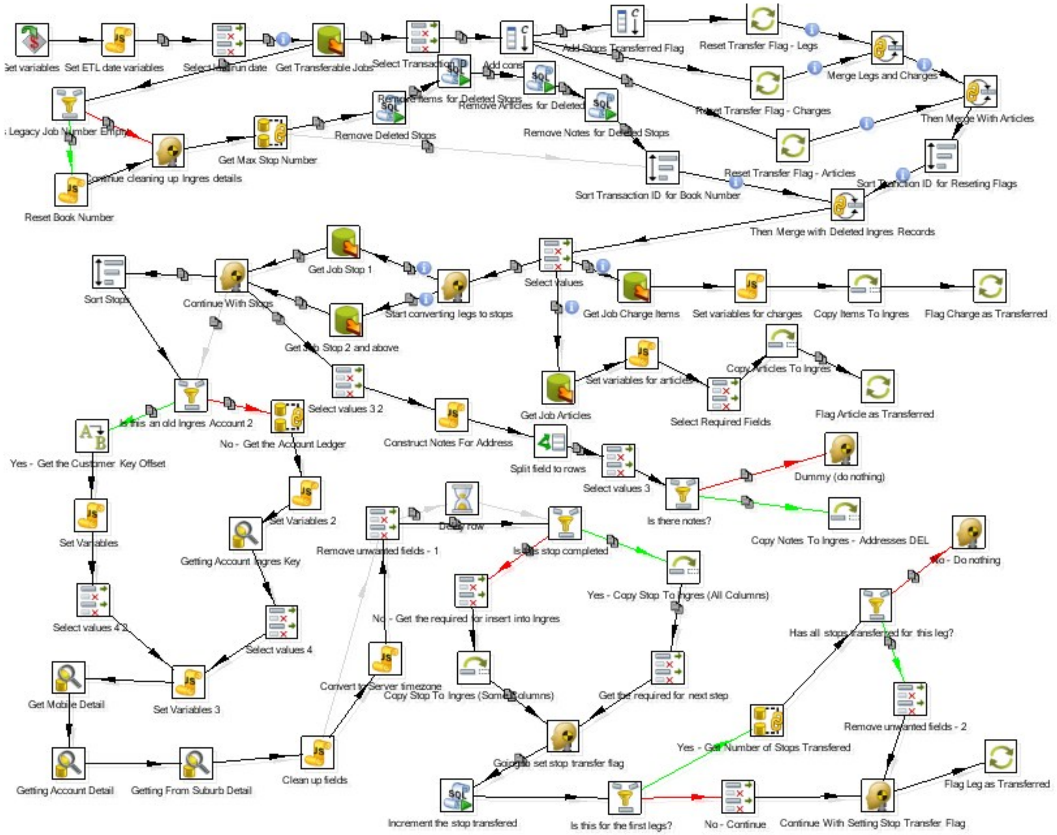
Testing philosophy

Testable solutions

Twineworks



Testable solutions



Set variables
Set ETL date variables
Select last run date
Get Transferable Jobs
Select Transaction ID
Add cons
Add Stops Transferred Flag
Reset Transfer Flag - Legs
Merge Legs and Charges
Reset Transfer Flag - Charges
Then Merge With Articles
Remove Items for Deleted Stops
Remove Articles for Deleted
Remove Notes for Deleted Stops
Legacy Job Number Empty
Get Max Stop Number
Remove Deleted Stops
Reset Transfer Flag - Articles
Sort Transaction ID for Reseting Flags
Continue cleaning up Ingres details
Reset Book Number
Sort Transaction ID for Book Number
Then Merge with Deleted Ingres Records
Sort Stops
Continue With Stops
Get Job Stop 1
Start converting legs to stops
Select values
Get Job Charge Items
Set variables for charges
Copy Items To Ingres
Flag Charge as Transferred
Get Job Stop 2 and above
Set variables for articles
Copy Articles To Ingres
Get Job Articles
Select Required Fields
Flag Article as Transferred
Is this an old Ingres Account 2
Select values 3 2
No - Get the Account Ledger
Yes - Get the Customer Key Offset
Construct Notes For Address
Split field to rows
Select values 3
Dummy (do nothing)
Is there notes?
Copy Notes To Ingres - Addresses DEL
Set Variables 2
Set Variables
Getting Account Ingres Key
Remove unwanted fields - 1
Delay row
Is this stop completed
Yes - Copy Stop To Ingres (All Columns)
No - Do nothing
No - Get the required for insert into Ingres
Has all stops transferred for this leg?
Select values 4 2
Select values 4
Get Mobile Detail
Set Variables 3
Convert to Server timezone
Copy Stop To Ingres (Some Columns)
Get the required for next step
Remove unwanted fields - 2
Yes - Get Number of Stops Transfered
Going to set stop transfer flag
Getting Account Detail
Getting From Suburb Detail
Clean up fields
Flag Leg as Transferred
Increment the stop transfered
Is this for the first legs?
No - Continue
Continue With Setting Stop Transfer Flag

Testable solutions

- Configuration management
  - configure all data sources/targets and paths through kettle variables or parameters
  - local environment (not in version control)
  - test environment (reference environment)
  - QA environment
  - production environment
- Resist the temptation to share configuration across environments
- Resist the temptation to introduce a hierarchy of environments

Testable solutions

- Define sub-systems/phases
  - define pre-requisites
    - data expected in certain sources
  - define outcomes
    - data written to certain sinks

- A sub-system/phase of the ETL process is responsible for a small set of related side-effects to happen

Testable solutions

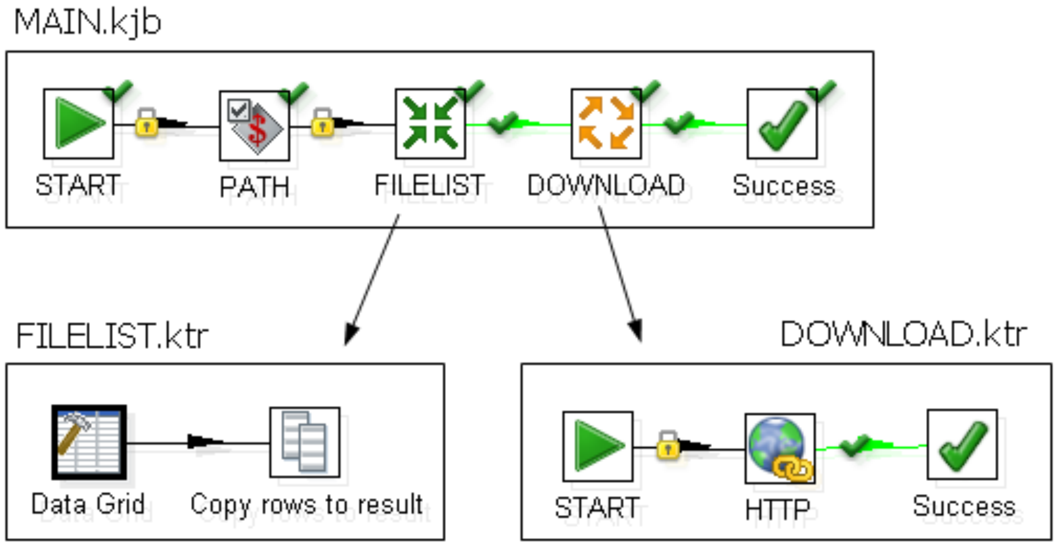
Twineworks

- Define entry points with a full functional contract.
- An entry point implements an application feature.

Testable solutions

Twine**works**

- Verify behavior of the entity you run directly



Minimize external deps

Twineworks

- run all data sources and targets locally or through dedicated instances of networked resources
- do not share anything you modify as part of the tests
- if team members cannot run the test suite locally, the test-suite will turn into a chore and a liability
- reproducible results: you can do it, your team mates can do it, and jenkins can do it

fixtures & helpers

Twineworks

- Data Fixtures
  - sets of test data, encoded in a convenient way, easily loaded into data sources and sinks
- JSON, CSV, SQL, XML, YAML
  - Use whatever is easiest to maintain for the team
- **Generate data fixtures** through parameterized scripts if you need to generate datasets with consistent relationships

fixtures & helpers

Twineworks

- File Fixtures
  - sets of test files acted upon during a run
- Maintain file fixtures separate from source location expected by ETL
- If fixture files are changed as part of the test, copy them to a temporary location before running tests
- Create a unique source location per test run, if the file location is shared (like sftp)

fixtures & helpers

- Helpers
  - utility code/etl of components reused to make tests about the what, not about the how
  - fixture loaders
  - assertion helpers
  - data comparison helpers

Declarative tests

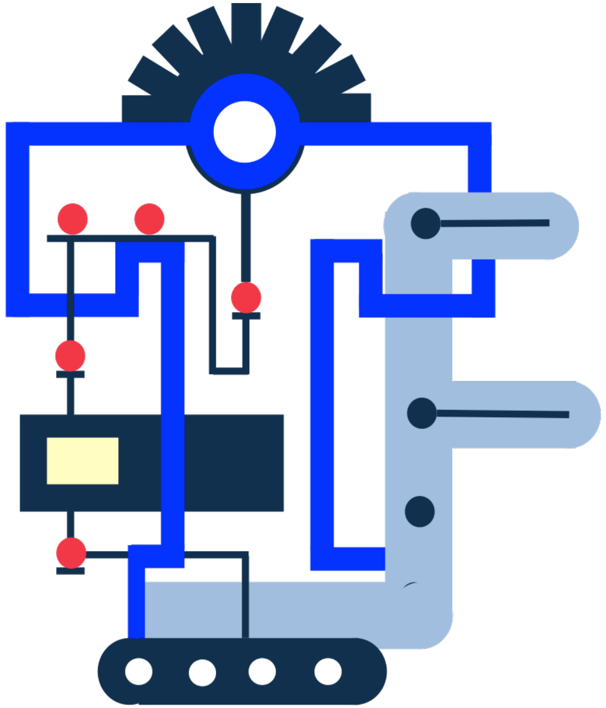
- A set of conventions, helpers, fixtures, and generic test runners that make it possible to define a test with minimal configuration and code.

Practical limitations

- Limited ability to provide test code that is
  - generic
  - data-driven

- metadata injection helps, but the increase in complexity is not trivial

Twineworks

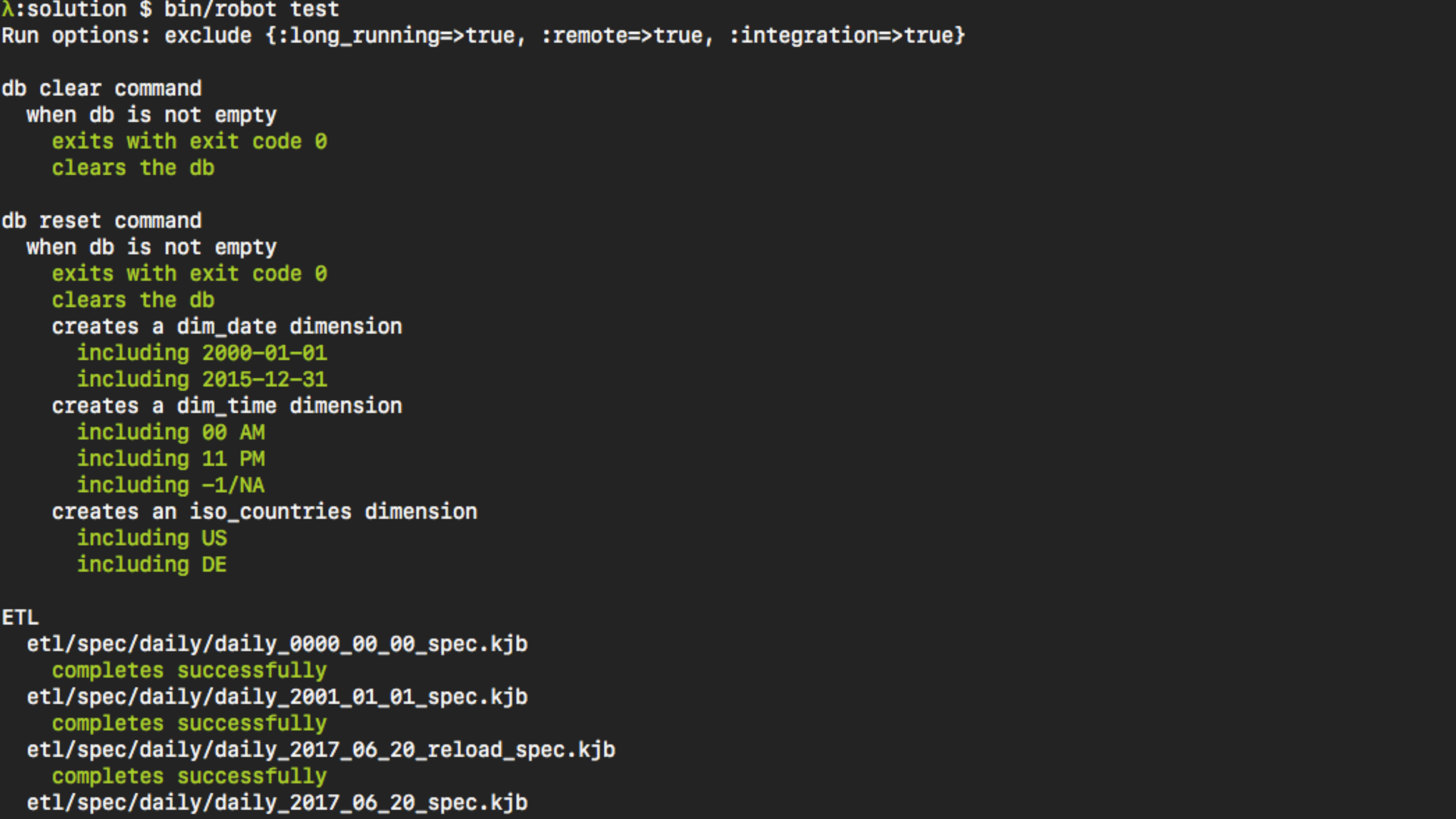


Scripting tests

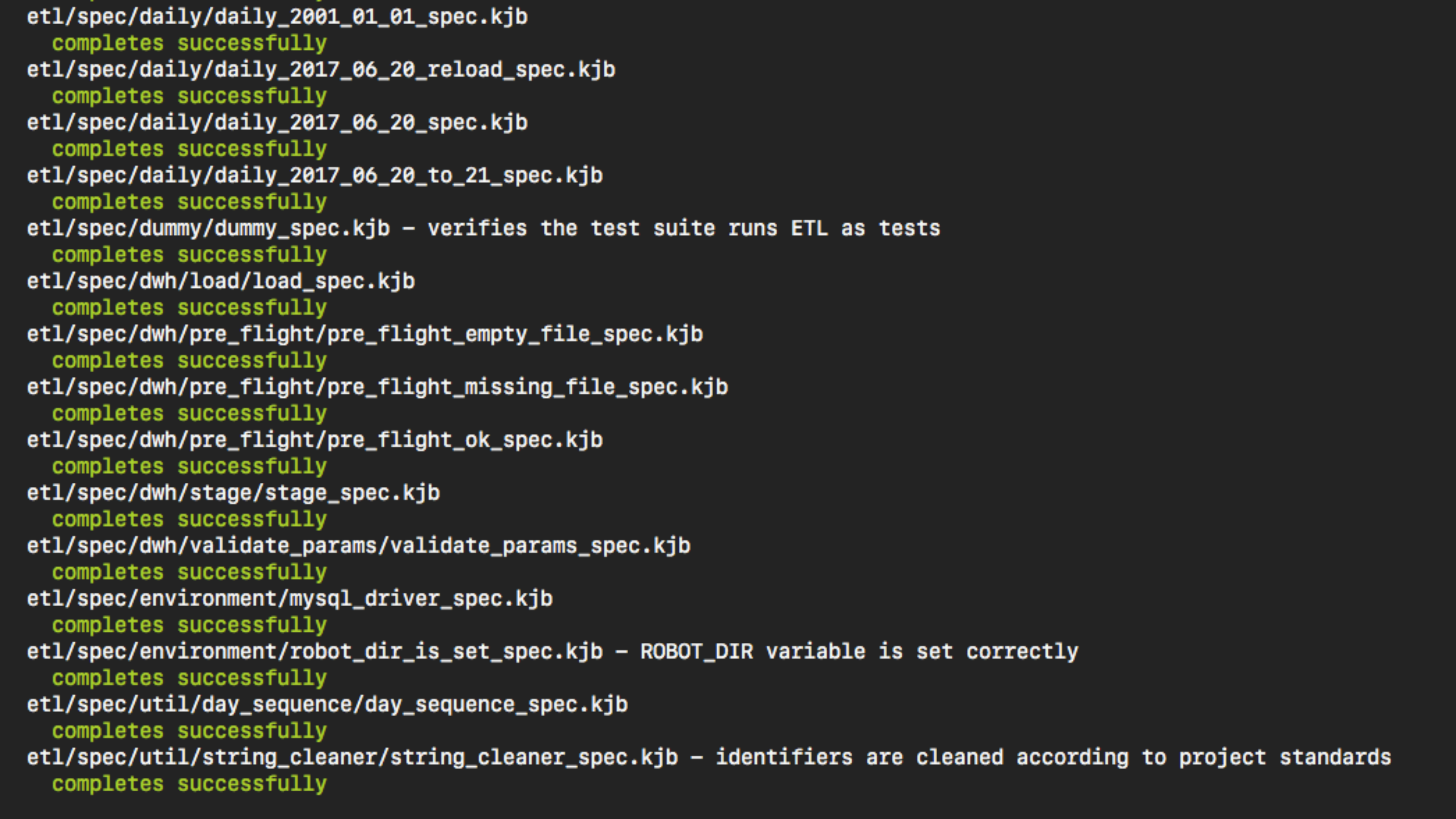
Test orchestration

Twineworks

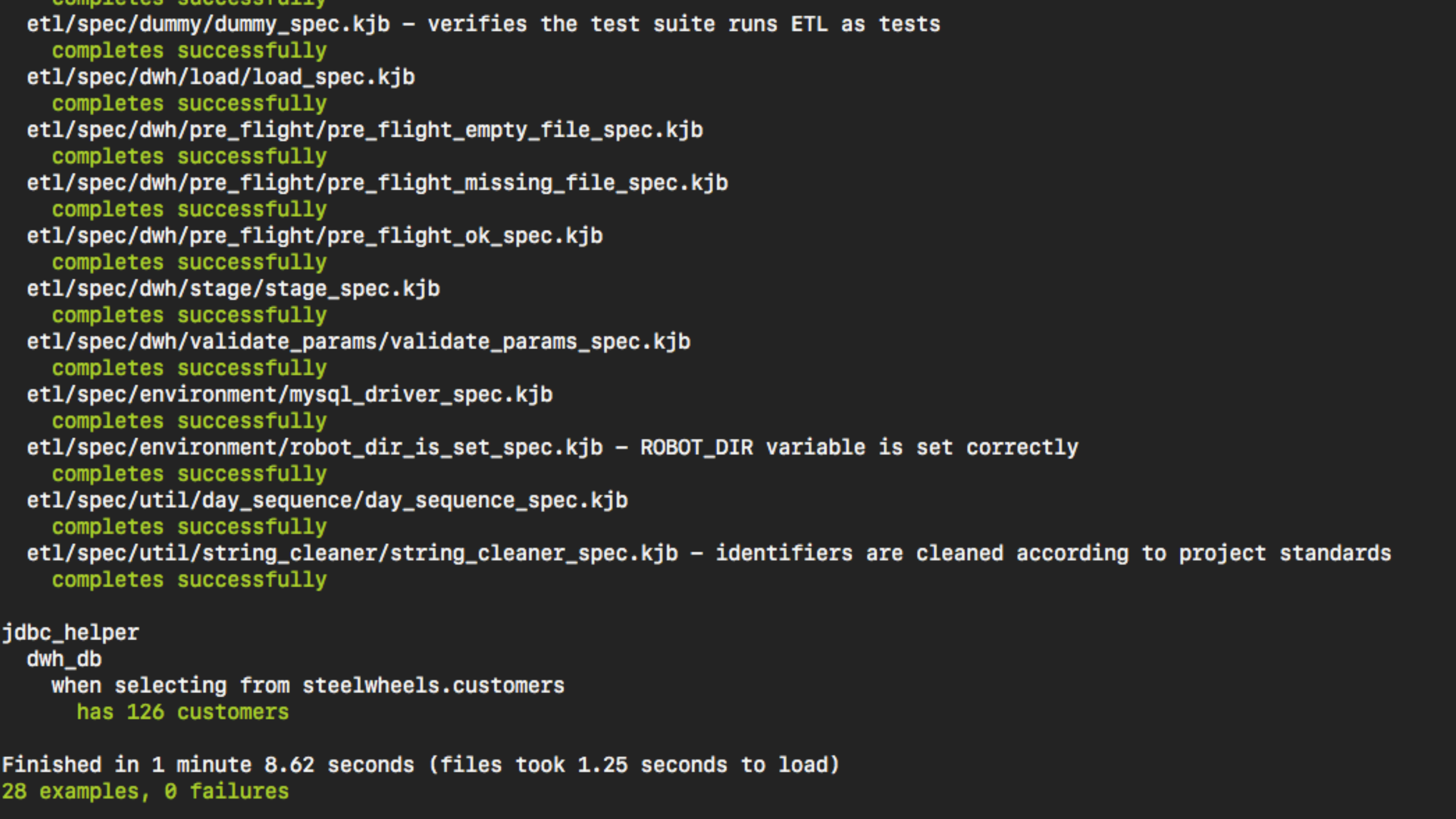
```
$ bin/robot test
```



```
λ:solution $ bin/robot test
Run options: exclude {:long_running=>true, :remote=>true, :integration=>true}

db clear command
  when db is not empty
    exits with exit code 0
    clears the db

db reset command
  when db is not empty
    exits with exit code 0
    clears the db
    creates a dim_date dimension
      including 2000-01-01
      including 2015-12-31
    creates a dim_time dimension
      including 00 AM
      including 11 PM
      including -1/NA
    creates an iso_countries dimension
      including US
      including DE

ETL
  etl/spec/daily/daily_0000_00_00_spec.kjb
    completes successfully
  etl/spec/daily/daily_2001_01_01_spec.kjb
    completes successfully
  etl/spec/daily/daily_2017_06_20_reload_spec.kjb
    completes successfully
  etl/spec/daily/daily_2017_06_20_spec.kjb
```

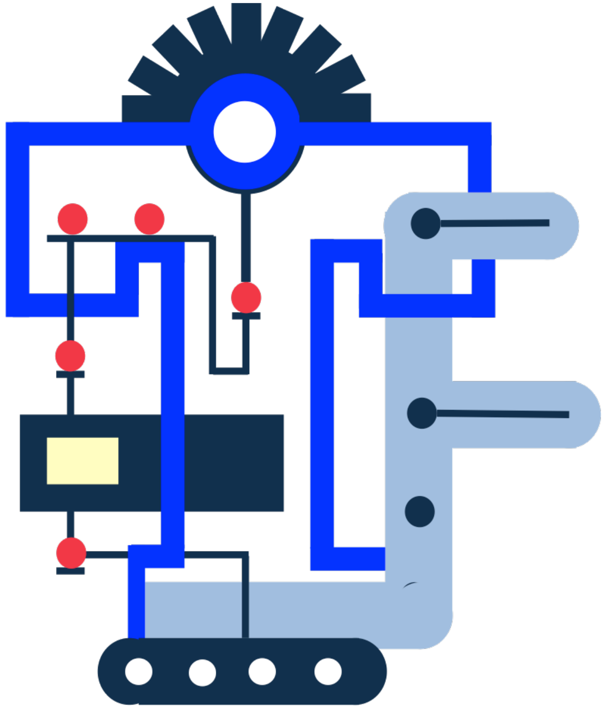


```
etl/spec/daily/daily_2001_01_01_spec.kjb
  completes successfully
etl/spec/daily/daily_2017_06_20_reload_spec.kjb
  completes successfully
etl/spec/daily/daily_2017_06_20_spec.kjb
  completes successfully
etl/spec/daily/daily_2017_06_20_to_21_spec.kjb
  completes successfully
etl/spec/dummy/dummy_spec.kjb - verifies the test suite runs ETL as tests
  completes successfully
etl/spec/dwh/load/load_spec.kjb
  completes successfully
etl/spec/dwh/pre_flight/pre_flight_empty_file_spec.kjb
  completes successfully
etl/spec/dwh/pre_flight/pre_flight_missing_file_spec.kjb
  completes successfully
etl/spec/dwh/pre_flight/pre_flight_ok_spec.kjb
  completes successfully
etl/spec/dwh/stage/stage_spec.kjb
  completes successfully
etl/spec/dwh/validate_params/validate_params_spec.kjb
  completes successfully
etl/spec/environment/mysql_driver_spec.kjb
  completes successfully
etl/spec/environment/robot_dir_is_set_spec.kjb - ROBOT_DIR variable is set correctly
  completes successfully
etl/spec/util/day_sequence/day_sequence_spec.kjb
  completes successfully
etl/spec/util/string_cleaner/string_cleaner_spec.kjb - identifiers are cleaned according to project standards
  completes successfully
```



```
    completes successfully
  etl/spec/dummy/dummy_spec.kjb - verifies the test suite runs ETL as tests
    completes successfully
  etl/spec/dwh/load/load_spec.kjb
    completes successfully
  etl/spec/dwh/pre_flight/pre_flight_empty_file_spec.kjb
    completes successfully
  etl/spec/dwh/pre_flight/pre_flight_missing_file_spec.kjb
    completes successfully
  etl/spec/dwh/pre_flight/pre_flight_ok_spec.kjb
    completes successfully
  etl/spec/dwh/stage/stage_spec.kjb
    completes successfully
  etl/spec/dwh/validate_params/validate_params_spec.kjb
    completes successfully
  etl/spec/environment/mysql_driver_spec.kjb
    completes successfully
  etl/spec/environment/robot_dir_is_set_spec.kjb - ROBOT_DIR variable is set correctly
    completes successfully
  etl/spec/util/day_sequence/day_sequence_spec.kjb
    completes successfully
  etl/spec/util/string_cleaner/string_cleaner_spec.kjb - identifiers are cleaned according to project standards
    completes successfully

jdbc_helper
  dwh_db
    when selecting from steelwheels.customers
      has 126 customers

Finished in 1 minute 8.62 seconds (files took 1.25 seconds to load)
28 examples, 0 failures
```

Twineworks



JRuby as a scripting language

Jruby

Twine**works**

Ruby language reference and std-lib documentation:

http://ruby-doc.org/

Jruby

Twineworks

Jruby is a ruby language implementation on the JVM

http://jruby.org/

- excellent Java interop features
- developed and maintained by RedHat

Jruby – A ruby primer

http://tryruby.org/

Jruby – interactive ruby

Twineworks

```
$ bin/robot jruby -S jirb
irb(main):001:0> 1+2
=> 3
irb(main):002:0> "Hello World"
=> "Hello World"
irb(main):003:0>
```

Jruby – variable assignment

Twineworks

```
irb> greeting = "Hello"
=> "Hello"
irb> greeting
=> "Hello"
```

Jruby – console output

Twineworks

```
irb> puts "Hello"
"Hello"
=> nil
```

Jruby – calling methods

Twine**works**

```
irb> "Hello".length
=> 5
irb> "Hello".length()
=> 5
irb> "Hello".gsub(/[aeiou]/, '*')
=> "H*ll*"
```

Jruby – conditionals

Twineworks

```
irb> if 5 > 3 then "Yes" else "No" end
=> "Yes"

irb> if 5 > 3
       "Yes"
     else
       "No"
     end
=> "Yes"
```

Jruby – strings

Twine**works**

```
irb> 'Hello World'
=> "Hello World"

irb> "Line1\nLine2"
=> "Line1\nLine2"

irb> puts "Line1\nLine2"
Line1
Line2
=> nil
```

Jruby – strings

```
irb> answer = 42
=> 42

irb> "answer is: #{answer}"
=> "answer is: 42"
```

Jruby – arrays

Twineworks

```
irb> []
=> []
irb> [12, 32, 42].size
=> 3
irb> items = [12, 32, 42]
=> [12, 32, 42]
irb> items.each do |x|
       puts x
     end
12
32
42
=> [12, 32, 42]
```

Jruby – arrays

```
irb> items.each {|x| puts x}
12
32
42
=> [12, 32, 42]
```

Jruby – arrays

```
irb> items.map {|x| x+100}
=> [112, 132, 142]

irb> [1, 2, 3, 4].reduce(0) {|a, x| a+x}
=> 10

irb> [1, 2, 3, 4].include? 0
=> false

irb> [1, 2, 3, 4].include? 3
=> true
```

Jruby – hashes

Twine**works**

```
irb> {}
=> {}

irb> {:first_name => "John", :last_name => "Doe"}
=> {:first_name => "John", :last_name => "Doe"}

irb> person = {:first_name => "John", :last_name => "Doe"}
=> {:first_name => "John", :last_name => "Doe"}

irb> person.each do |k, v|
       puts "#{k} is #{v}"
     end
first_name is John
last_name is Doe
=> {:first_name=>"John", :last_name=>"Doe"}
```

Jruby – hashes

Twineworks

```
irb> person.keys
=> [:first_name, :last_name]

irb> person.values
=> ["John", "Doe"]

irb> person.to_a
=> [[:first_name, "John"], [:last_name, "Doe"]]
```

Jruby – environment vars

Twineworks

```
irb> ENV
=>{"ROBOT_DIR"=>"/Users/slawo/Desktop/etl-
testing/solution", "XPC_FLAGS"=>"0x0” …}

irb> ENV.keys
=> ["ROBOT_DIR", "XPC_FLAGS",
"TERM_PROGRAM", "HOME", …]
```

Jruby – Ruby quick reference

Twineworks

Ruby cheat sheets

https://github.com/marcusvmsa/cheatsheets/tree/master/ruby

Jruby – running a ruby script

Twine**works**

```
$ bin/robot jruby *file*
```

```
$ bin/robot jruby ruby/hello.rb
Hello World!
```

Jruby – write a ruby script

Twineworks

Write a ruby script that outputs all environment variables to the console.
Each variable goes on a sepearate line in the format: VAR=VALUE

```
$ bin/robot jruby ruby/environment_variables.rb
GEM_HOME=/Users/slawo/Desktop/etl-
testing/solution/bin/jruby/gem_home
TERM_PROGRAM_VERSION=388.1
USER=slawo
DISPLAY=/private/tmp/com.apple.launchd.TTWKUYpXJx/org.macosforge.xq
uartz:0
HOME=/Users/slawo
…
```

Jruby – defining methods

Twine**works**

```
irb> def add(a=0, b=0)
          a+b
     end
=> nil

irb> add
=> 0

irb> add 7
=> 7

irb> add 7, 9
=> 16

irb> add 7, 9, 6
ArgumentError: wrong number of arguments calling `add` (3 for 2)
```

Jruby – command line args

Twineworks

```
irb> ARGV
=> []

irb> ARGV.size
=> 0
```

Jruby – Dir

Twineworks

```
irb> Dir.pwd
=> "/Users/slawo/Desktop/etl-testing/solution"

irb> Dir.glob("./**/*.csv")
=> ["./bin/jruby/gem_home/gems/diff-lcs-1.3/spec/fixtures/ds1.csv",
 "./bin/jruby/gem_home/gems/diff-lcs-1.3/spec/fixtures/ds2.csv",
 "./data/in/2017/06/2017-06-18.csv",
 "./data/in/2017/06/2017-06-19.csv",
 "./data/in/2017/06/2017-06-20.csv",
 "./data/in/2017/06/2017-06-21.csv",
 "./etl/init/date/fixed_date_holidays.csv",
 "./etl/init/locale/iso_countries.csv",
 "./etl/spec/util/string_cleaner/string_cleaner_spec.csv”,
…]
```

Jruby – write a ruby script

Twine**works**

Write a ruby script that outputs all file paths below a given directory that end in a given prefix.

The script prints a usage message if the number of arguments is not two.

```
$ bin/robot jruby ruby/find_in_dir.rb etl .kjb
etl/daily.kjb
etl/dwh/validate_params.kjb
etl/dwh/load/load.kjb
etl/dwh/pre_flight/pre_flight.kjb
...
$ bin/robot jruby ruby/find_in_dir.rb etl
usage: find_in_dir.rb dir suffix
```

Jruby – metaprogramming

Twine**works**

```
irb> class Hash
       def keys_reversed
         keys.reverse
       end
     end
=> nil

irb> {:a => "a", :b => "b"}.keys
=> [:a, :b]

irb> {:a => "a", :b => "b"}.keys_reversed
=> [:b, :a]
```

Jruby – Gems

Twineworks

```
irb> "hello".blank?
NoMethodError: undefined method `blank?' for "hello":String

irb> require "active_support/all"
=> true

irb> "hello".blank?
=> false

irb> "".blank?
=> true
```

http://guides.rubyonrails.org/active_support_core_extensions.html

Jruby – Gems

Twineworks

```
# jruby/Gemfile

source 'http://rubygems.org'
gem 'gson'      # json processing
gem 'nokogiri'  # xml processing
gem 'rspec’     # testing framework
gem 'rspec-its'
gem 'rspec_junit_formatter' # rspec formatter for xml output on CI
gem 'jdbc-helper’ # convenient jdbc queries
gem 'iniparse’    # convenient ini parsing
gem 'activesupport', '4.2.8' # language core extensions
```

Jruby – write a ruby script

Add the cowsay gem https://github.com/johnnyt/cowsay to the project by modifying Gemfile and running

$ bin/jruby/install-gems

```
Write a ruby script that has the avatar say the first argument, using the optional second argument for the template.
```

```
$ bin/robot jruby ruby/cowsay.rb 'A good day to you!'
```

```
 ____________________
| A good day to you! |
 --------------------
      \   ^__^
       \  (oo)\_______
          (__)\       )\/\
              ||----w |
              ||     ||
```

Jruby – write a ruby script

Twineworks

```
$ bin/robot jruby ruby/cowsay.rb 'A good day to you!' bunny

 ____________________
| A good day to you! |
 --------------------
  \
   \   \
        \ /\
        ( )
      .( o ).
```

Jruby – write a ruby script

Twineworks

Write a ruby script that, given the path to a ktr or kjb file, outputs the description entered in the job or transformation properties dialog, if any.

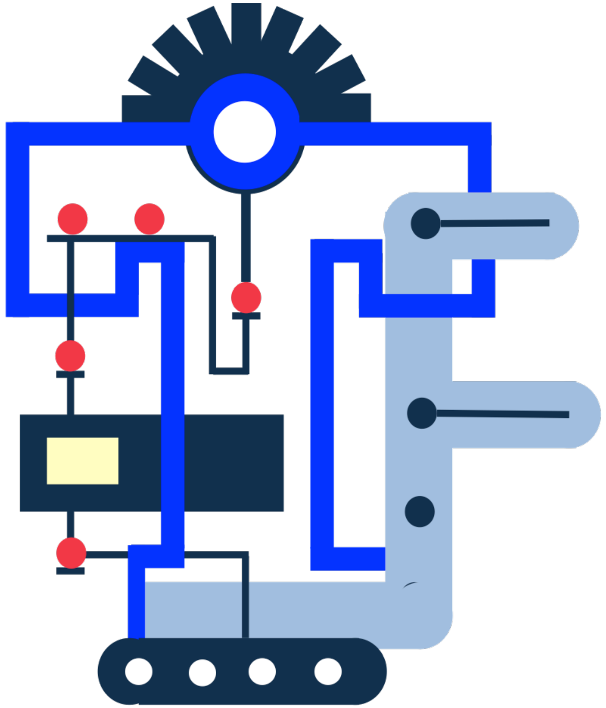
```
$ bin/robot jruby ruby/meta.rb etl/util/string_cleaner.ktr
identifiers are cleaned according to project standards
```

```
$ bin/robot jruby ruby/meta.rb etl/daily.kjb
main entry point for the robot application
```

```
$ bin/robot jruby ruby/meta.rb missing_file
file does not exist: missing_file
```

```
$ bin/robot jruby ruby/meta.rb bin/load/help.txt
unknown file format: bin/load/help.txt
```

Twineworks



Orchestrating the test suite with rspec

Rspec

Rspec is a testing framework for ruby

http://rspec.info/

https://relishapp.com/rspec

Rspec

Twine**works**

```
$ bin/robot test
```

- invokes rspec
- rspec loads **spec/spec_helper.rb**
- **spec/spec_helper.rb** sets configuration options
- includes all ruby files in **spec/support**
- ruby files in **spec/support** define any helper methods that the actual tests use
- rspec traverses the **spec** folder looking for files whose names end in **_spec.rb** and loads them as **tests**

Rspec - a spec file

Twineworks

```
describe "db clear command" do

  describe 'when db is not empty' do

    before :all do
      dwh_db.load_fixture 'spec/fixtures/steelwheels/steelwheels.sql'
      @out, @err, @status = execute "bin/robot db clear"
    end

    # print output of commands if anything failed
    after(:each) do |example|
      if example.exception
        puts @out
        puts ""
        puts @err
      end
    end

    it 'exits with exit code 0' do
      expect(@status).to eq 0
    end

    it "clears the db" do
      expect(dwh_db.query("SHOW TABLES").to_a.uniq.length).to eq 0
    end

  end

end
```

Rspec – describe blocks

Twineworks

Describe blocks are grouping tests into related units. They can be nested.

```
describe ”some context" do

    ...

end
```

Reference documentation for the basic structure of a test file: https://relishapp.com/rspec/rspec-core/v/3-4/docs/example-groups/basic-structure-describe-it

Rspec – before and after

Twine**works**

- Describe blocks can contain **before** and **after** hooks.
- If any tests within a describe block are run, the corresponding enclosing before and after hooks are executed as defined.
- The hooks can be scoped to run before/after **all** or **each** test.

```
describe "subsystem foo" do
  before :all do
    @my_var = "foo"
  end
  ...
end
```

Rspec – before and after

Twineworks

In ETL testing the **before** hook is typically used to execute some ETL and capture any immediate output and exit codes into **instance** variables.

Instance variables are prepended with an **@** sign, and are available in before, after and example blocks.

https://www.relishapp.com/rspec/rspec-core/v/3-4/docs/hooks/before-and-after-hooks

Rspec – assertions

The actual assertions happen within **it** blocks, that describe expected behavior.

```
describe "subsystem foo" do

  before :all do
    @my_var = "foo"
  end

  it "is equal to 'foo'" do
    expect(@my_var).to eq "foo"
  end

end
```

Assertions are made through **expect**, which takes a value and supports a set of matchers. The matcher used above is **eq**.

Rspec – assertions

Twine**works**

Matcher reference documentation:

http://www.relishapp.com/rspec/rspec-expectations/v/3-4/docs/built-in-matchers

Rspec – test execution

Twineworks

Rspec runs in two phases

Phase 1: collects tests, recording the structure as given by the describe blocks.

Phase 2: executes tests, using several command line options to limit the execution to just the tests fulfilling certain criteria, like having a certain substring in its name, or being tagged with certain keywords.

Rspec – test execution

Twine**works**

Run only tests containing the word ‘clear’ in their name or enclosing describe blocks:

```
$ bin/robot test  --example 'clear'
```

Run only tests tagged ‘long_running’:

```
$ bin/robot test  --tag 'long_running'
```

Run only tests in **spec/commands**

```
$ bin/robot test  spec/commands
```

Rspec – test execution

Twineworks

Reference documentation on how to tag a group of tests with metadata
https://relishapp.com/rspec/rspec-core/v/3-4/docs/metadata/user-defined-metadata

Reference documentation for rspec command line options
https://relishapp.com/rspec/rspec-core/v/3-4/docs/command-line

Rspec helpers

Twineworks

spec/support/spec_helpers.rb

```
def execute cmd
  stdin, stdout, stderr, thread = Open3.popen3(cmd)
  [stdout.read, stderr.read, thread.value.to_i]
end
```

Runs cmd as a shell command and returns an array returning
[stdout, stderr, exit_code]

Rspec helpers

Twine**works**

spec/support/spec_helpers.rb

```
def dwh_db
  ...
end
```

Runs a jdbc database object offering convenient access to the **dwh** database as per https://github.com/junegunn/jdbc-helper

The database connection is created on demand, and closes automatically when rspec ends.

Rspec helpers

Twine**works**

spec/support/spec_helpers.rb

```
def dwh_db
   ...
end
```

In addition

`dwh_db.load_fixture(path)` allows loading a sql or json fixture file

`dwh_db.reset()` triggers **$ bin/robot db reset**

Running jobs as rspec tests

Twine**works**

spec/etl/etl_spec.rb

Recursively traverses **etl/spec** looking for files whose names end in **_spec.kjb**, and dynamically generates a **describe** and **it** block for it.

Hence all such job files are part of the test suite.

Test the load phase

Create a spec

`spec/load/load_spec.rb`

The spec verifies that **`/etl/dwh/load/load.kjb`**

- loads data from stage_traffic
- has aggregated the records for same date, time, user and service
- has calculated the user_tag by transforming it as expected
- has filled the service table as expected

Use **`/fixtures/load/2017-06-21.json`** to populate the **stage_traffic** table.

Test compliance

Create a spec

`spec/environment/mysql_driver_spec.rb`

The spec verifies that the file **${ROBOT_PDI_HOME}/lib/mysql-connector-java-5.1.42-bin.jar** exists.

It also verifies that there is no other file matching pattern **mysql-connector-java-*-bin.jar** in that folder.

Declarative Tests

Twine**works**

Create a spec
`spec/daily/daily_spec.rb`

Recursively traverse **spec** looking for folders whose names end in **_daily_dec**, and dynamically generate **describe** and **it** blocks for them.

Each **_daily_dec** folder contains a folder **input**, a file **run_dates.txt**, and a file **expected.csv**.

`The spec starts by clearing the contents of the db.`

`Then, for each date in` **`run.dates,`** `the spec executes`

```
$ bin/robot job run etl/daily.kjb -param:DATA_DATE=<date> -param:DATA_DIR=path/to/*_daily_dec/input
```

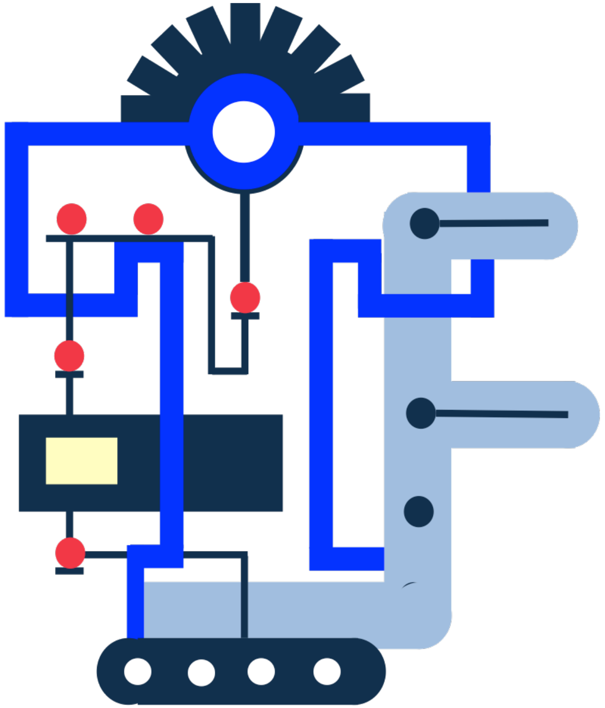
`Then the spec asserts that the state of the traffic tables is equal to` **`expected.csv`**`.`

Declarative Tests

Convert all ETL functional tests from **etl/spec/daily** to the declarative format.

Place them in **spec/daily_dec**.

Twineworks



Continuous testing with Jenkins

Jenkins

Twineworks

Jenkins

Jenkins is a continuous integration server.

It's basic role is to run the test suite and build any artifacts upon changes in version control.

Example server:
http://ci.pentaho.com/

Installing Jenkins

Get **jenkins.war** from http://mirrors.jenkins.io/war/2.66/

$ java -jar jenkins.war

Jenkins places all its configuration in **~/.jenkins**

Installing Jenkins

Twineworks

Jenkins

Install the following plugins:

AnsiColor
build-timeout-plugin
Credentials Binding Plugin
Email Extension Plugin
Git plugin
HTML Publisher plugin
Timestamper
Workspace Cleanup Plugin
Junit plugin

Some plugins will install as dependencies

Installing Jenkins

Twineworks

Jenkins

Configure Jenkins to use a single executor.

Placing the ETL in git

Twineworks

```
# create a bare repository
mkdir -p /home/slawo/Desktop/etl-project.git
cd /home/slawo/Desktop/etl-project.git
git init --bare --shared=group

# make the solution folder a git repository
cd /home/slawo/Desktop/solution
git init
git add .
git commit -m 'initial commit'

# potentially have to set up commit identity
git config --global user.email "slawomir.chodnicki@twineworks.com"
git config --global user.name "Slawomir Chodnicki"

# let local repository know where remote is
git remote add origin /home/slawo/Desktop/etl-project.git

# push to bare repository
git push -u origin master
```

An environment for Jenkins

Twine**works**

Duplicate your **local** environment for jenkins as **test**

`Use {WORKSPACE} as placeholder for project directory in`

- **`environments/test/.kettle/kettle.properties`**
- **`environments/test/environment`**

Database configuration

Twineworks

```
$ mysql -u root

CREATE DATABASE dwh_jenkins;

GRANT ALL PRIVILEGES ON dwh_jenkins.* TO etl@'%' identified by 'password';
GRANT ALL PRIVILEGES ON dwh_jenkins.* TO etl@'localhost' identified by 'password';

FLUSH PRIVILEGES;
```

Main test job

Twineworks

```
# bail on errors
set -e

# set environment to use
export ROBOT_ENVIRONMENT=test

# set workspace directory in config files
sed -i.bak s:{WORKSPACE}:${WORKSPACE}:g environments/test/.kettle/kettle.properties
sed -i.bak s:{WORKSPACE}:${WORKSPACE}:g environments/test/environment

# install dependencies
bin/jruby/install

# run test suite
bin/robot test
```

Ensure Jenkins can run tests

Twine**works**

Get the test suite to pass on Jenkins, fixing configuration issues as they arise.

Don’t forget to properly **commit** and **push** your changes.

Jenkins artefacts

Twineworks

Jenkins

Configure Jenkins to use the **rspec-integration.xml** test report file.

Jenkins artefacts

Configure Jenkins to **archive** all artefacts on successful builds.

Jenkins nightly

Jenkins

Configure Jenkins to create a nightly build in a separate job.

Jenkins concurrent builds

Twine**works**

Jenkins

What would it take to enable multiple executors?

Jenkins pipelines

Twineworks

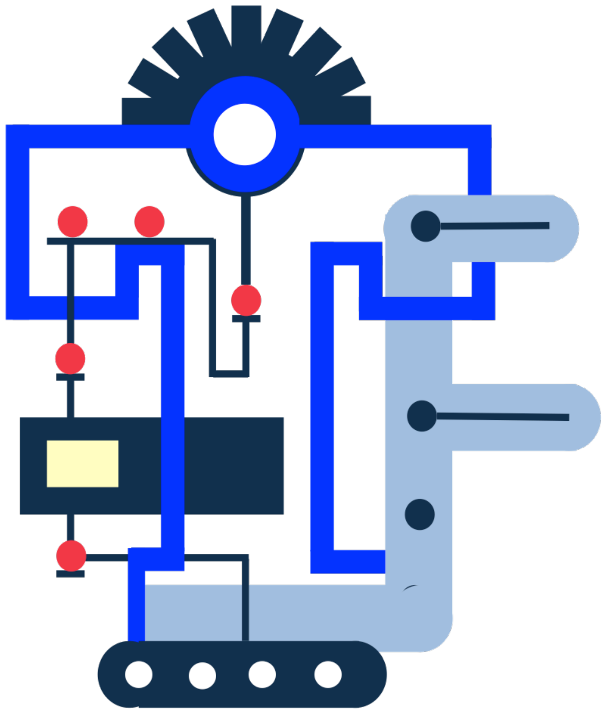
Jenkins

Jenkins **pipelines** implement workflows.

The Blue Ocean project provides a sensible UI for pipelines.

https://jenkins.io/projects/blueocean/

Twineworks



Simple deployment from git

Deploying from git

Twine**works**

Tag two releases as

$ git tag releases/2017-06-21_r1 *commit_hash*

$ git tag releases/2017-06-21_r2 *commit_hash*

And push to shared repo

$ git push origin releases/2017-06-21_r1

$ git push origin releases/2017-06-21_r2

List remote tags

$git ls-remote --tags origin

Example deployment scripts

Twineworks

```
.
|-- conf                # configuration
|-- extract.sh          # extracts a tag
`-- activate.sh         # symlinks a tag as active
```

Example deployment scripts

Twine**works**

Extract and activate **2017-06-21_r1**

```
$ ./extract.sh 2017-06-21_r1
$ ./activate.sh 2017-06-21_r1
```

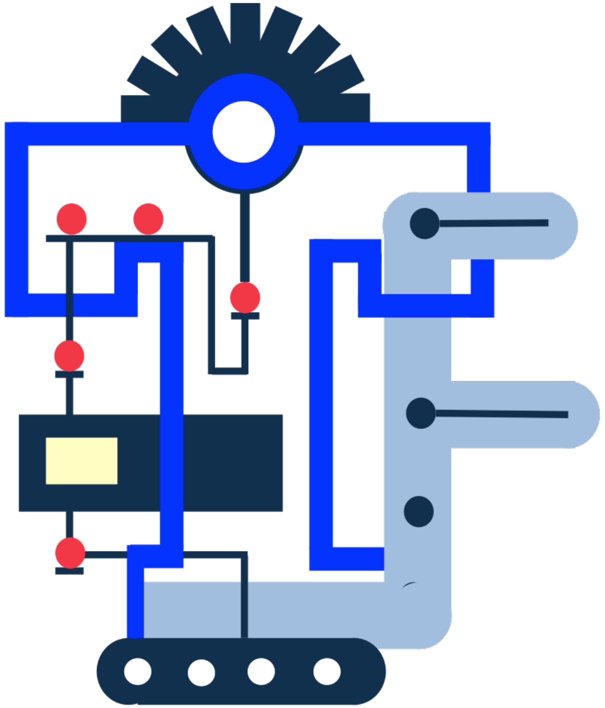
Extract and activate **2017-06-21_r2**

```
$ ./extract.sh 2017-06-21_r2
$ ./activate.sh 2017-06-21_r2
```

Rollback to **2017-06-21_r1**

```
$ ./activate.sh 2017-06-21_r1
```

Twineworks



Database schema migration

Manual migration scripts

Twineworks

Manually perform DB schema migrations by creating a SQL script or kjb to migrate to the next version.

After deployment run

```
$ bin/robot db run etl/migrations/release_panda/add_tables.sql
```

Or if it is a kjb

```
$ bin/robot job run etl/migrations/release_panda/panda_migration.kjb
```

Test and dry run migrations on QA like any other code.

Avoid destructive changes, i.e. rename columns instead of dropping them. Drop after confirmed lasting release success.

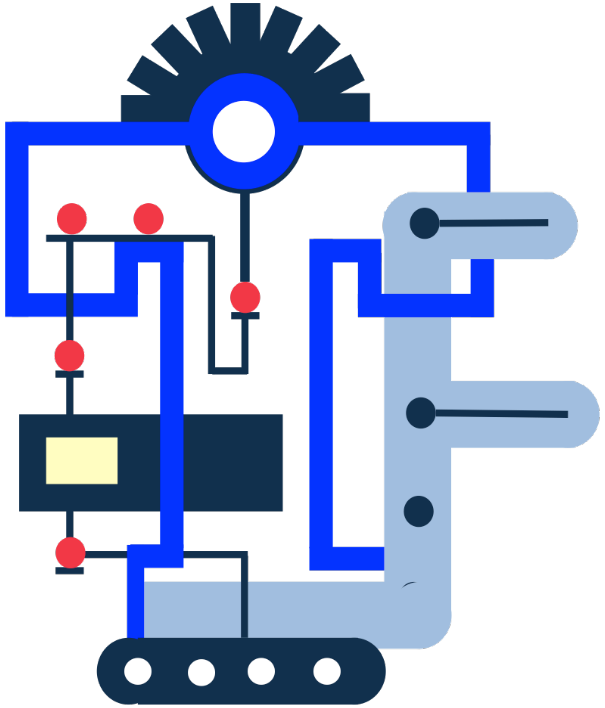
Managed migration scripts

Twineworks

Flyway automates DB schema migrations by storing a version number in the schema and maintaining a set of SQL scripts or java code to migrate from one version to the next.

https://flywaydb.org/

Twineworks



Thank you!