CMDB Driven by Perl

Road to a Perl "driven" Configuration Management Database

Jens Rehsack

Niederrhein Perl Mongers

The Perl Conference in Amsterdam

Motivation

Progress

Progress isn't made by early risers. It's made by lazy men trying to find easier ways to do something.

(Robert A. Heinlein)

Motivation

Efficiency

Business success is because of Perl. It enables us to deliver right solutions in days instead of months.

*(*Elizabeth Mattijsen*)*

Goal

Automation

Full flavoured systems management

- Installation without Administrator interaction
- Control sensors and alarming
- Ensured system state by actual-theoretical comparison
- Faster reaction in emergency cases by organized component moving
- Have an up-to-date "Operation Handbook" as well as archiving them

Begin with reporting

Begin with reporting

In the beginning was the (Installation-)Report

- Technical Sales defined an XML Document for Change Requests and Status Reports
- Based on work of forerunner a 70% solution could be delivered
- Document Definition lacks entity-relations
- Document Definition misses technolgy requirements

⇒ Appears to be a dead end

From reporting to . . .

Mind the goal

Alice: Would you tell me, please, which way I ought to go from here?
The Cheshire Cat: That depends a good deal on where you want to get to.

*(*Lewis Carroll*)*

Where do I begin

To write the workflow how great Perl 5 can be

- The project was in a state where a developer created a particular Report based on the existing snapshot.
- This solution did not maintain an abstraction layer for gathered data - every time when the report needs an extension, an end-to-end (snapshot to XML-Tag) enhancement had to be created.
- Changes shall be deployed from the same report format as installations are reported.
- We have to be able to say at any moment what is operated on the platform.

Baby Steps

Improve knowledge

Based on identified issues the first goal had to be to identify all entities and their relations together

Surrounded

Problem: The entire platform was completely unstructured

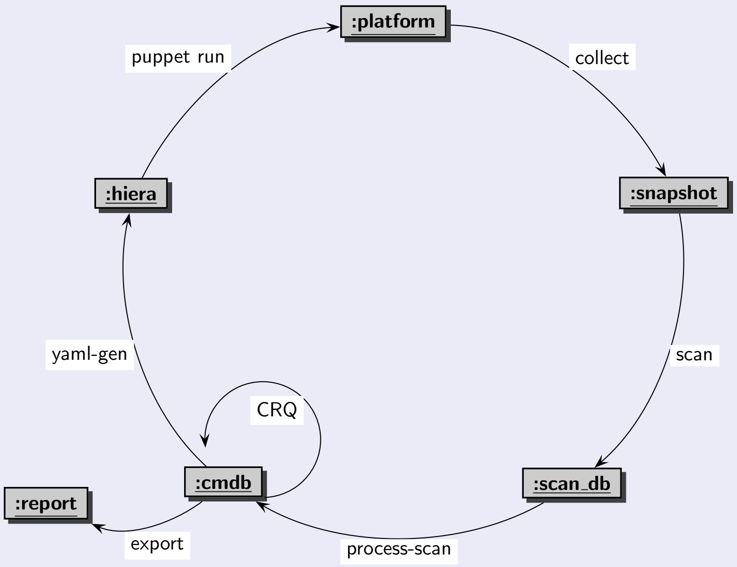
Baby Steps

Multiple Beginnings

- The already known "(Installation-)Report"
- Platform Snapshot (SCM Repository of selected configuration files)
- Puppet Classes (without *Hiera*) mixed with *Configuration Items* (within *Hiera*) and prepared configuration files (unsupervised)

  Hiera is Puppet's built-in key/value data lookup system. By default, it uses simple YAML or JSON files, although one can extend it to work with almost any data source.

Circle in the Sand



Technical Concerns

Rough

- Collecting platform parameters (to query them in structured way)
- Identify coherences of *Configuration Items* (CI)
- Define a data model
- Define technical requirements

Practical Concerns

Practical

- Validity of `CI`'s
- Limits of our `CI`'s
- Data ownership of `CI`'s
- Methods to persist `CI`'s
- Methods to access `CI`'s
- Permission management

Impossible Things

Impossible Things

Alice laughed. "There's no use trying," she said: "one can't believe impossible things."

"I daresay you haven't had much practice," said the Queen. "When I was your age, I always did it for half-an-hour a day. Why, sometimes I've believed as many as six impossible things before breakfast." (Lewis Carroll)

The Fool with a Tool

Try again

So we closed our eyes, took a deep breath (multiple times) and looked around for tools to store serialized data and read in structured way ...

Tool Time

MongoDB	allows easy storing in any format - but lacks structured querying dedicated entities (configuration items)
Data Files	delegate relationship handling completely to business logic
AnyData2	gotcha - allows reading most confusing stuff and could be queried in structured way

. . . is still a Fool

Volatile Structure

- Persist structured data using SQLite
- Define a data model representing existing relations
- Develop `AnyData2::Format` classes representing defined ER (*Entity Relationship*) model
- Develop simple MOP inside this AnyData2 instance to manage attributes vs. columns
- Glue everything together using SQL

The entire ER model remains a moving target

Abstraction Layer

...of configured components

- Focus the goal to know what is operated
- Depth first search over all component configuration files
- Identify relationships (remember: there is no operation model at all)
- Clean up configuration when no reasonable relationships can be resolved or relationships are conflicting

Moo in practice

Moo in practice

It appears that the tools helping to do safe IoT device updating are the same tools helping to coordinate CI determining:

`MooX::Cmd`	helps separating concerns
`MooX::ConfigFromFile`	helps contribute "divine wisdom"
`MooX::Options`	allow overriding "divine wisdom" by "individual wisdom"
`MooX::Log::Any`	feeds `DBIx::LogAny`

Moo in background

Moo in background

- Manage database connections based on concerns
- Manage `CI` structures based on relations
- Manage Web-API integration

Craziness

Crazy

I'm not crazy. My reality is just different than yours.

The Cheshire Cat

(Lewis Carroll)

Harmonization

Harmonize Craziness

- Practically any administrator had a different background regarding to the platform components thus a different picture of their relationships
- EPIC battles leads to common craziness
- ER model analysis sessions uncovered holes in picture

Civilized

March Hare: Have some wine.
(Alice looked all round the table, but there was nothing on it but tea.)
Alice: I don't see any wine.

March Hare: There isn't any.
Alice: Then it wasn't very civil of you to offer it.
March Hare: It wasn't very civil of you to sit down without being invited.
(Lewis Carroll)

Adding a Goal

CentOS 5 ends its maintenance

- Many of existing tools need to be upgraded
- Upgraded tools don't support existing hacks anymore
- Existing hacks must be replaced by a reasonable configuration structure
- Same problem like the report format:
  - neither the ER model of platform components nor issues of platform where known
  - nor cared about

Self Protection

Delegation

We learned from mistakes of past:

- No responsibility taken for filling weird puppet templates
- No external data will be managed
- No precompiled/puzzled resources are prepared

⇒ ER model of `CMDB` is presented via *RESTful API*

Scan completed

- Early implementation of above mentioned *RESTful API* run against *ScanDB*
- *ScanDB* represents just a view of the configuration snapshot
- There is no future, nor past
- Time for *CMDB* to enter the stage

Customers . . .

```
CREATE TABLE customer_t
(
      customer_id INTEGER PRIMARY KEY
    -- entity stuff
    , customer_name VARCHAR(80) UNIQUE NOT NULL
    -- cmdb stuff
    , valid_from DATETIME NOT NULL
    , valid_to DATETIME
    , modified_at DATETIME NOT NULL
    , modified_by VARCHAR(32) NOT NULL
);
```

- primary key and global identifier for this data type
- the payload of this data type, automatically indexed
- CMDB manages history and updates using these columns

VPN links to customers . . .

```
CREATE TABLE vpn_link_t
(
      vpn_link_id INTEGER PRIMARY KEY
    -- entity stuff
    , customer_id INTEGER NOT NULL
    , vpn_link_type VARCHAR(12)
    , customer_net VARCHAR(64) UNIQUE NOT NULL
    , services_net VARCHAR(64) UNIQUE NOT NULL
    -- cmdb stuff
    , valid_from DATETIME NOT NULL
    , valid_to DATETIME
    , modified_at DATETIME NOT NULL
    , modified_by VARCHAR(32) NOT NULL
    -- FK
    , FOREIGN KEY (customer_id) REFERENCES customer_t(customer_id)
      ON UPDATE CASCADE ON DELETE CASCADE
);
```

- refer the customer
- support Cisco, Juniper, Paolo Alto, . . .
- networks must be unique or network admins kill you

Moo Interception

```
package Foo::Role::Database::CMDB;
use Moo::Role;
requires "log";

has cmdb => (
    is       => "ro",
    required => 1,
    handles  => "Foo::Role::Database",
    isa      => sub {
        _INSTANCE_OF($_[0], "Foo::Helper::CMDB") and $_[0]->DOES("Foo::Role::Dat
          and return;
        die "Insufficient initialisation parameter for cmdb";
    },
    coerce => sub {
        _HASH($_[0]) and return Foo::Helper::CMDB->new(%{$_[0]});
        $_[0];
    },
);
```

- role can be consumed by any class needing access to CMDB
- transform hash initializer into object

Hard work

```
package Foo::Helper::CMDB;
use Moo; extends "Foo::Helper::DatabaseClass";
has config_tables => (is => "lazy", ...);
has history_tables => (is => "lazy", ...);
around deploy => sub { ...
my @tables = @{$self->config_tables};
foreach my $tbl (@tables) {
  my @hist_coldefs =
    map { my $default = defined $_->[4] ? " DEFAULT $_->[4]" : "";
      $pure_cols{$_->[1]} ? ("$_->[1] $_->[2]$default")
        : ("old_$_->[1] $_->[2]$default", "new_$_->[1] $_->[2]$default")
    } @table_info;
}
unshift @hist_coldefs, "${base_name}_hist_id INTEGER PRIMARY KEY";
my $hist_defs = join("\n    , ", @hist_coldefs);
my $hist_tbl = <<EOCHT;
CREATE TABLE ${base_name}_hist (
      ${hist_defs}
);
EOCHT
```

- that are all tables with trailing _t in their names
- create history table for each config table
- memoizing old and new values on updating payload

Hard work (continued) - INSERT

```
my $new_cols = join(", ", map { $pure_cols{$_} ? $_ : "new_$_" }
                grep { !$skipped{$_} } @table_cols);
my $new_vals = join(", ", map {"NEW.$_"} grep { !$skipped{$_} } @table_cols);

my $new_trgr = <<EONT;
CREATE TRIGGER new_${base_name}_row AFTER INSERT ON ${base_name}_t
BEGIN
  INSERT INTO ${base_name}_hist ($new_cols)
  VALUES ($new_vals);
END;
EONT
```

⇒ ON INSERT fill history rows without filling "OLD_" columns

Hard work (continued) - UPDATE

```
my (@updt_cols, @rfrs_cond, @updt_vals);
foreach my $colnm (grep { !$skipped{$_} } @table_cols) {
  my @lcd = $pure_cols{$colnm} ? $colnm : ("old_${colnm}", "new_${colnm}");
  push @updt_cols, @lcd;
  push @rfrs_cond, $pure_cols{$colnm}
    ? _cmp_if_nullable("existing.${colnm}", "NEW.${colnm}")
    : (_cmp_if_nullable("existing.old_${colnm}", "OLD.${colnm}"),
       _cmp_if_nullable("existing.new_${colnm}", "NEW.${colnm}"));
  push @updt_vals, $pure_cols{$colnm}?("NEW.$colnm"):("OLD.$colnm","NEW.$colnm"
}
my $updt_cols = join(", ", @updt_cols);
my $updt_refreshed_cols = join(", ", map { "refreshed.$_" } @updt_cols);
my $updt_vals = join(", ", @updt_vals);
```

⇒ Prepare for a bit complexer trigger

⇒ Distinguish between real updates and just "touches"

Hard work (continued) - UPDATE

```
my $updt_trgr = <<EONT;
CREATE TRIGGER updt_${base_name}_row AFTER UPDATE ON ${base_name}_t
BEGIN
  INSERT OR REPLACE INTO ${base_name}_hist (${base_name}_hist_id, $updt_cols)
  VALUES (
    (SELECT MAX(existing.${base_name}_hist_id) ${base_name}_hist_id
     FROM ${base_name}_hist existing WHERE $updt_refreshed_cond),
    $updt_vals);
END;
EONT
```

⇒ ON UPDATE create history (`INSERT`) rows with "OLD_" and "NEW_" columns
except nothing changes (`REPLACE`)

"UPSERT"

```
MERGE INTO tablename USING table_reference ON (condition)
  WHEN MATCHED THEN
  UPDATE SET column1 = value1 [, column2 = value2 ...]
  WHEN NOT MATCHED THEN
  INSERT (column1 [, column2 ...]) VALUES (value1 [, value2 ...]);
```

SQLite

- Unsupported by SQLite
- `INSERT OR REPLACE` deletes before insert

→ Kills `UPDATE` Trigger

Perl helps out

```
$self->cmdb->upsert( customer_t => {
  customer_name => "Foo Enterprises", });
$self->cmdb->upsert( vpn_link_t => {
  customer_name => "Foo Enterprises",
  vpn_link_type => "Juniper",
  customer_net  => "10.116.47.8/29",
  services_net  => "10.126.47.8/29" } );
```

SQL created . . .

```
INSERT OR IGNORE INTO vpn_link_t (
  customer_id, vpn_link_type, customer_net, services_net, modified_by
) VALUES (
  (SELECT customer_id FROM customer_t WHERE customer_name=?),
  ?, ?, ?, ?);
UPDATE vpn_link_t SET vpn_link_type=?, customer_net=?, services_net=?,
  modified_by=?, touched_at=CURRENT_TIMESTAMP
WHERE changes()=0 AND customer_id=(
  SELECT customer_id FROM customer_t WHERE customer_name=?);
```

Known limitations

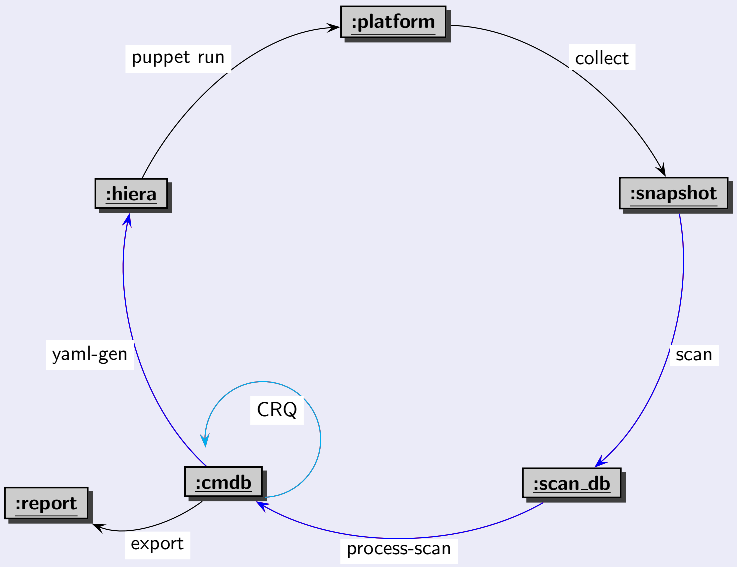
- Restricted to CMDB
- Refuse updates of identifying columns (UNIQUE constraints)
- WHERE clause derived from UNIQUE constraints

CMDB to Hiera

YAML Generator

- Development team read via *RESTful API* the theoretical configuration set
- *Hiera* `YAML` files are written
- Additional exports are managed via *Hiera*
- Puppet classes are rewritten to understand new ER model

Circle closed



Goals reached

→ Actual-theoretical comparison done via processing scan database

→ Unmaintainted installation via cronjob possible

→ Reaction in emergency cases by organized component moving done multiple times

→ Monitoring, sensors, alarming open

Conclusion

Can a programming language save a life

- Yes, it can - but here it saves our business

Thank You For Listening

Questions?

Jens Rehsack <rehsack@cpan.org>
Cologne