Spacewalk on PostgreSQL

Jan Pazdziora
Principal Software Engineer
Satellite Engineering, Red Hat

Developer Conference 2011
12th February 2011
Brno, Czech Republic

What is Spacewalk?

- System management system, with WebUI and XMLRPC API.
- The project is the upstream for Red Hat Network Satellite product (and SuSE Manager).
- Satellite went open source in June 2008.
- Written in Java, Python, and Perl.
- 700+ thousand lines of code (and 2M+ lines of markup).
- `http://spacewalk.redhat.com/`
- `https://fedorahosted.org/spacewalk/`

The database backend

- Spacewalk traditionally runs on Oracle database.
- To make the solution completely open source, PostgreSQL support is under way.
- We want to support both databases. This is not a "rewrite and forget Oracle" effort.

- We will go through the history and state of the port, will list issues that we hit, show examples, and hopefully inspire you to contribute or use our experience in migration of your legacy applications.

PostgreSQL port timeline

- June 2008: the work started.
  - Contributions from EnterpriseDB and Tom Lane.
- August 2009: Spacewalk 0.6 had the bits in.
  - Required manual steps to even setup Spacewalk.
  - No tracking of changes in the Oracle land.
- Late 2009: Work slowed down.
- Summer 2010: Revisited.
  - Made the installation and setup pass automatic.
  - Focus on sustainability: made it easy to see what had changed in Oracle.

PostgreSQL port timeline (cont'd)

- November 2010: Spacewalk 1.2.
  - Released with support for PostgreSQL.
    - And with porting guide, kudos to Lukáš Zapletal.
  - Feature subset.
  - Stunned silence. Virtually no contributions.
- February 2011: Spacewalk 1.3.
  - Minor fixes.
  - You could upgrade from 1.2.

The PostgreSQL port as of today

- Spacewalk 1.2 on PostgreSQL supports:
  - content sync with satellite-sync and spacewalk-repo-sync
  - rhnpush
  - client registration
  - yum operations
- Spacewalk 1.3 added:
  - Spacewalk Proxy activation
  - the database schema upgradable

Package dependencies

- We had to split out the dependencies on Oracle-specific packages (`perl-DBD-Oracle` and friends).
- Spacewalk is now installed either as

```
yum install spacewalk-oracle
```

or

```
yum install spacewalk-postgresql
```

Helper CLI

- When populating and upgrading database schema, we use the native SQL codes (maybe generated from common syntax by chameleon).
- At the same time, the connect information to the database is already stored in configuration file.
- We've created `spacewalk-sql` to do the right thing with both backends, connecting to Oracle with `sqlplus` or to PostgreSQL with `psql`.

Schema definition

- Some things are the same on both Oracle and PostgreSQL: `CREATE VIEW`.
- Some things are similar with subtle differences: `CREATE TABLE` where the column types differ, `varchar2` vs. `varchar`, `date` vs. `timestamp`.
  - We used tool chameleon.
  - Reformatted the definition, lost named constraints.
  - Just use sed.
- Need to have "compatibility" objects: `dual`, `numtodsinterval`, `nvl`, `nextval`.

Schema definition (cont'd)

- Some things need to be written again: packages vs. schemas, procedures, triggers.
  - Need to ensure that the "second class citizen" sources stay in sync with the "first class".

```
-- oracle equivalent source sha1 2284fa9b8b...
```

  - And enforce it (in .rpm build time).
- Schema upgrades — common sources or duplicate sources. Again, ensure consistency.

Examples

https://fedorahosted.org/spacewalk/wiki/PostgreSQLPortingGuide

Functions nvl, nvl2, decode, ...

- You can either create the functions in PostgreSQL.
- Or replace with `CASE WHEN ... THEN ... END`.

Select without FROM, and dual

- In Oracle, there is always table `dual` with single record.
- In PostgreSQL, `SELECT` without `FROM` clause can be used instead.
- We just create `dual` on PostgreSQL.

Examples (cont'd)

Syntax sugar

- The AS keyword is needed for column aliases in PostgreSQL.
- And subqueries need to be named with aliases there.

Outer joins

- Oracle syntax: `FROM TAB1, TAB2 WHERE TAB1.ID = TAB2.TAB1_ID (+)`
- ANSI syntax: `FROM TAB1 LEFT OUTER JOIN TAB2 ON TAB1.ID = TAB2.TAB1_ID`

Examples (cont'd)

Date types and arithmetics

- If your application already uses timestamps, good.
- If you use `date` type and `sysdate`: beware that `date` in Oracle has precision up to seconds, in PostgreSQL up to days. Type `date` in Oracle has arithmetics, adding 1 adds one day.
- We have replaced `sysdate` with `current_timestamp`, but with great care.
- We also had to craft functions for `date` to seconds (epoch) conversion, and back.

Examples (cont'd)

Triggers

- If the `tg_op` in PostgreSQL is `INSERT`, we cannot even touch `old`.

Procedure invocation from Java

- Stored procedures need to be called as

```
{ call procx(:user_id); }
```

instead of

```
begin procx(:user_id); end;
```

Examples (cont'd)

Anonymous PL/(pg)SQL

- Invocation of anonymous procedural SQL blocks are not supported by PostgreSQL at all.
- The ideal solution would be to rewrite them all to functions and procedures.
- We've addressed the issue by adding a hack to our database layer when SHA1 of the PL/pgSQL body is computed and it is used as part of procedure name which gets created on the fly (if it does not yet exist) and then called.

Examples (cont'd)

`DISTINCT` and `ORDER BY`

- PostgreSQL does not like

```
select distinct a, b, c, label
from rhnchannel
order by upper(label)
```

- We used subselect.

`rownum`

- Oracle has a special pseudocolumn `rownum` which can be used to limit resultset, similar to PostgreSQL's `LIMIT`.
- We've changed the logic.

Examples (cont'd)

User types

- When referencing fields of composite types, parentheses are needed:

```
select data.id, (data.evr).version from data
```

- Methods are not supported by PostgreSQL, so we just use global functions

```
select evr_t_as_vre_simple(data.evr) from data
```

instead of

```
select data.evr.as_vre_simple() from data
```

Examples (cont'd)

nextval

- In Oracle, `nextval` is a method which is called on the sequence object:

```
select rhnpackage_seq.nextval from dual
```

  In PostgreSQL, global function with sequence name is used instead:

```
select nextval('rhnpackage_seq')
```

  We just created compatibility function in Oracle to achieve the same.

Closing remarks

- Spacewalk with PostgreSQL installs and runs.
- You can even convert your existing Spacewalk installation with Oracle backend to use PostgreSQL.
- Come and submit patches.
- Or use as inspiration for your migrations.

- Thank you for your attention.