



Analysing Debian packages with Neo4j

Norbert Preining

<http://www.preining.info/>



Tokyo, Japan

<http://www.accelia.net/>



debian

Debian Developer

2017-10-19

Self introduction



- ▶ Logician by education, 20+ years in research on Theoretical Computer Science and Mathematical Logic at various universities
- ▶ Since one year working at Accelia Inc. (CDN/IT Service) on security and machine learning
- ▶ Debian Developer since about 20 years, mainly responsible for T_EX related packages
- ▶ Developer of T_EX Live, main author of T_EX Live Manager and installer

Self introduction



- ▶ Logician by education, 20+ years in research on Theoretical Computer Science and Mathematical Logic at various universities
- ▶ Since one year working at Accelia Inc. (CDN/IT Service) on security and machine learning
- ▶ Debian Developer since about 20 years, mainly responsible for T_EX related packages
- ▶ Developer of T_EX Live, main author of T_EX Live Manager and installer
- ▶ Complete beginner with graph databases ;-)

Why graph database?



- ▶ Preparing a recommender system for potential clients
- ▶ Natural way to represent the available data
- ▶ Learning something new
- ▶ Highly non-hierarchical data in graph database (many examples have a clear hierarchical structure)
- ▶ Reasonable sized (not too small) and meaningful content

Why graph database?



- ▶ Preparing a recommender system for potential clients
- ▶ Natural way to represent the available data
- ▶ Learning something new
- ▶ Highly non-hierarchical data in graph database (many examples have a clear hierarchical structure)
- ▶ Reasonable sized (not too small) and meaningful content
- ▶ Hip!(!?)

Overview



- ▶ Quick introduction to Debian
- ▶ Packages in Debian
- ▶ Ultimate Debian Database
- ▶ Representing packages as a graph (“Database schema”)
- ▶ Conversion from UDD to Neo4j
- ▶ Sample queries and visualizations
- ▶ Concluding remarks



Introduction to Debian



- ▶ Open source Linux distribution
- ▶ Developed (mostly) by volunteers
- ▶ Lots of offspring (e.g. Ubuntu)
- ▶ Strict license requirements (DFSG)
- ▶ About 31000 source packages building about 82000 binary packages

Debian releases



Stable

officially released, security support and updates

Debian releases



Stable

officially released, security support and updates

Testing

preparation for the next stable

Debian releases



Stable

officially released, security support and updates

Testing

preparation for the next stable

Development (sid)

entrance point for all packages, main development place

Debian releases



Stable

officially released, security support and updates

Testing

preparation for the next stable

Development (sid)

entrance point for all packages, main development place

Experimental

what it says, for testing, often used during pre-release freeze

Debian releases



Stable

officially released, security support and updates

Testing

preparation for the next stable

Development (sid)

entrance point for all packages, main development place

Experimental

what it says, for testing, often used during pre-release freeze

Other releases: point releases for stable, oldstable, historic releases

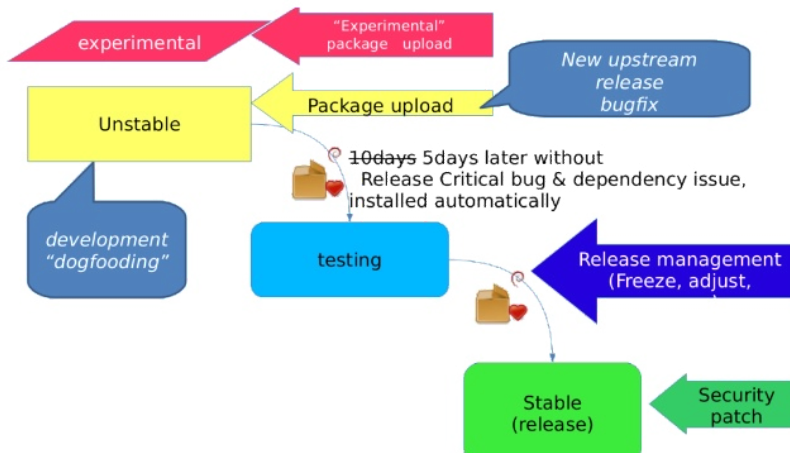


Image by Youhei Sasaki (CC-NC-SA)

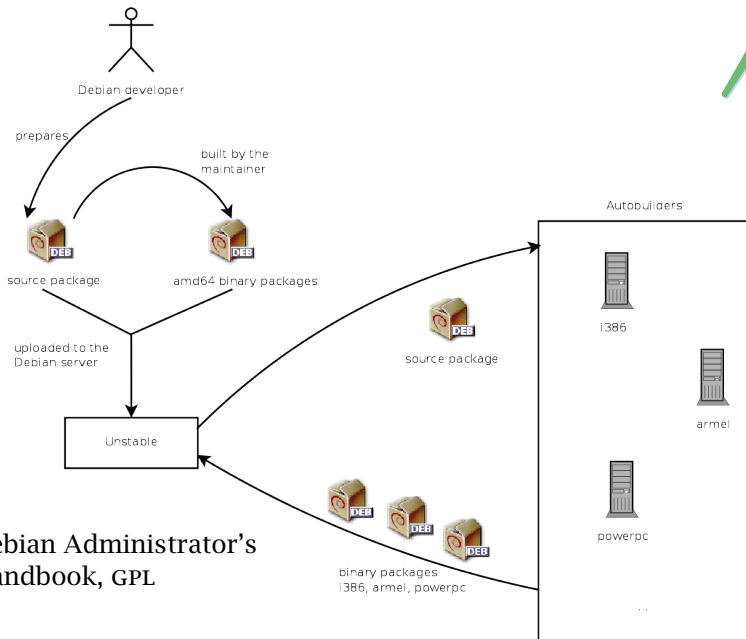


Packages in Debian

Packages



- ▶ source packages and binary packages
- ▶ Developer uploads source package (and his own's arch binary package, or source-only upload)
- ▶ other architectures are built by auto-builders
- ▶ upload is included in unstable (or rejected)



Debian Administrator's
Handbook, GPL

Versions of packages in Debian



- ▶ current versions in sid, testing, stable, oldstable, security releases
- ▶ intermediate versions that did not end up in a release

Example

asymptote package:

oldstable: 2.31-2, stable: 2.38-2, testing and sid: 2.41-2

other versions uploaded to unstable: 2.35-1, 2.35-2, 2.37-1, 2.38-1, 2.41-1, ...

Version numbers



`[epoch:]upstream_version[-debian_revision]`

Examples

asymptote 2.41-2:

upstream_version: 2.41

debian_release: 2

Version numbers



`[epoch:]upstream_version[-debian_revision]`

Examples

`asymptote 2.41-2:`

`upstream_version: 2.41`

`debian_release: 2`

`musixtex 1:1.20.ctan20151216-4:`

`epoch: 1`

`upstream_version: 1.20.ctan20151216`

`debian_release: 4`

Components of a package



- ▶ Maintainer: who is responsible
- ▶ Uploaders: who is allowed to upload packages
- ▶ Section, Priority: relevant for structuring the huge set of packages
- ▶ Version
- ▶ dependency declarations
- ▶ lots of further fields

Some caveats



- ▶ one source package can build many different binary packages
- ▶ the names of source package and binary package are not necessary the same (necessarily different when building multiple binary packages)
- ▶ binary packages of the same name (but different version) can be built from different source packages

Dependencies



for source packages

Build-Depends, Build-Depends-Indep, Build-Depends-Arch,
Build-Conflicts, Build-Conflicts-Indep, Build-Conflicts-Arch

for binary packages

Depends, Pre-Depends, Recommends, Suggests, Enhances,
Breaks, Conflicts

various formats of dependencies

Relation: pkg

Relation: pkg (<< version)

Relation: pkg | pkg

Relation: pkg [arch1 arch2]



Ultimate Debian Database UDD

The UDD



Imports data from a variety of sources:

- ▶ packages and source files, from both Debian and Ubuntu
- ▶ bugs from the Debian BTS
- ▶ popularity contest
- ▶ history of uploads
- ▶ history of migrations
- ▶ lintian (conformance check tool)
- ▶ orphaned packages
- ▶ debtags, carnivore, Ubuntu bugs, NEW queue, DDTP translations, ...



The UDD schemata

<https://udd.debian.org/schema/>

UDD schemata



- ▶ highly de-normalized
- ▶ good example of grown-over-time schemata
- ▶ lots of duplication without connections

UDD schemata



- ▶ highly de-normalized
- ▶ good example of grown-over-time schemata
- ▶ lots of duplication without connections
- ▶ a pleasure for any SQL fetishist ;-)



Can we put the UDD into a Graph Database?

Entities: first steps: source and binary packages

source builds binary relation

Use different node types for (versioned) source and binary package; link increasing versions



Entities: first steps: source and binary packages

source builds binary relation

Use different node types for (versioned) source and binary package; link increasing versions

unversioned dependencies

Use different node types for unversioned and versioned source and binary packages



Entities: first steps: source and binary packages

source builds binary relation

Use different node types for (versioned) source and binary package; link increasing versions

unversioned dependencies

Use different node types for unversioned and versioned source and binary packages

Node and relations (for now)

Nodes: sp (source package), vsp versioned source packages, bp (binary package), vbp versioned binary package

```
vsp -[:is_instance_of]-> sp
vbp -[:is_instance_of]-> bp
sp -[:builds]-> bp
vbp -[:next]-> vbp
vsp -[:next]-> vsp
```



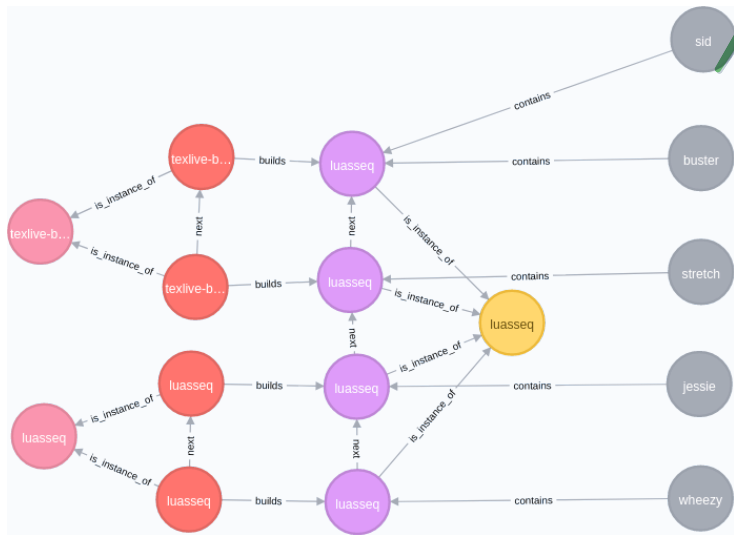




Register the binary packages that are included in a suite (release):

Node type: suite

```
suite -[:contains]-> vbp
```

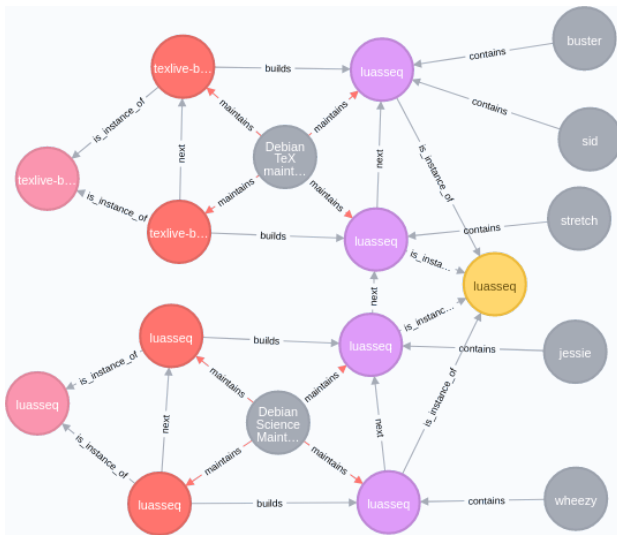




Register the maintainers of binary and source packages:

Node type: mnt

```
mnt -[:maintains]-> vbp  
mnt -[:maintains]-> vsp
```



Dependencies

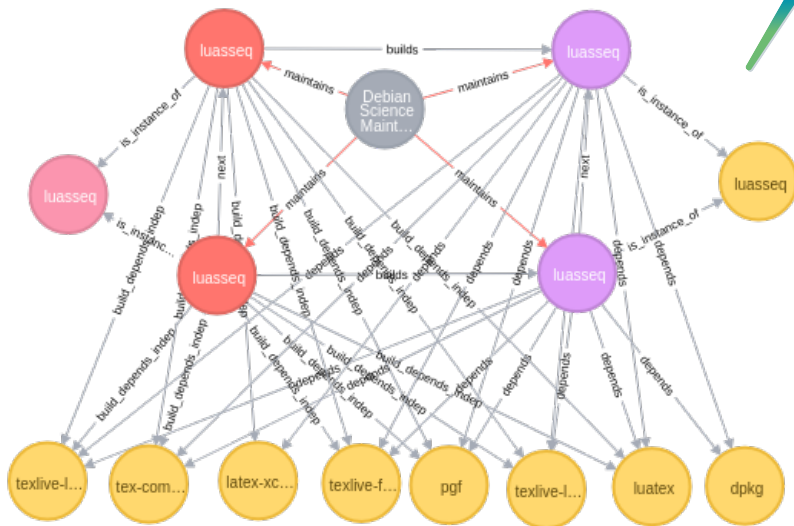


Current state: dependency is represented as relation between a versioned (source/binary) package and *unversioned* binary package with additional properties (type of relation, version number)

```
vbp -[:depends reltype: TYPE, relversion:  
VERS]-> bp
```

Where TYPE is one of <<, <=, ==, >=, >>.

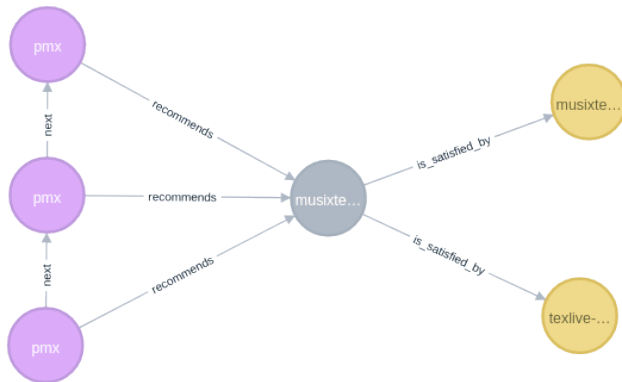
If it is an *unversioned* relation TYPE is none, and relversion is 1.



Alternative dependencies



Add a new node type `altdep` and a new relation `is_satisfied_by`.



name of the `altdep`:

```
musixtex (>= 1:0.98-1) | texlive-music
```




Summary of Nodes and Relations

Nodes and relations



Nodes and attributes

- ▶ mnt: name, email
- ▶ bp, sp, suite, altdeps: name
- ▶ vbp, vsp: name, version

Relations and attributes

- ▶ breaks, build_conflicts, build_conflicts_indep, build_depends, build_depends_indep, conflicts, depends, enhances, is_satisfied_by, pre_depends, provides, recommends, replaces, suggests:
Attributes: reltype, relversion
- ▶ builds, contains, is_instance_of, maintains, next:
no attributes

Number of entities



Nodes

suite: 28, mnt: 3510, altdeps: 8852, sp: 31889, bp: 154808,
vsp: 81567, vbp: 247419

Total: 528079

Relations

breaks: 58141, build_conflicts: 3377,
build_conflicts_indep: 32, build_depends: 587051,
build_depends_indep: 103097, builds: 234897,
conflicts: 45424, contains: 358341, depends: 1843653,
enhances: 6294, is_instance_of: 328986,
is_satisfied_by: 22517, maintains: 328986, next: 211840,
pre_depends: 11998, provides: 109171, recommends: 96187,
replaces: 71560, suggests: 117655

Total: 4539206



Conversion from UDD to Neo4j

Conversion step 1: Getting the data



- ▶ UDD has a public mirror:
`public-udd-mirror.xvm.mit.edu`
- ▶ Postgresql DB, use Perl, DBI::PG to get the various tables

Conversion step 2: Into Neo4j

My first try was generating Cypher statements ... lots of them.



Conversion step 2: Into Neo4j



My first try was generating Cypher statements ... lots of them.

That was not really good ;-)

Conversion step 2: Into Neo4j



My first try was generating Cypher statements ... lots of them.

That was not really good ;-)

Use `neo4j-import` tool

- ▶ generate for each node/relation a csv with ids
- ▶ run `neo4j-import`, takes a few seconds!

Conversion step 2: Into Neo4j



My first try was generating Cypher statements ... lots of them.

That was not really good ;-)

Use neo4j-import tool

- ▶ generate for each node/relation a csv with ids
- ▶ run neo4j-import, takes a few seconds!

```
$ neo4j-import ...  
...  
IMPORT DONE in 10s 608ms.  
Imported:  
  528073 nodes  
  4539206 relationships  
  7540954 properties  
Peak memory usage: 521.28 MB
```

How to generate node/relation csv?



- ▶ Perl program parsing the csv files from psql
- ▶ generates a huge hash with all information (in fact more than currently evaluated)
- ▶ for each item generated a unique UUID
- ▶ generates the necessary csv files



Sample queries

Checking build-deps



Find all packages in Jessie that build depends on some version of `tex-common`:

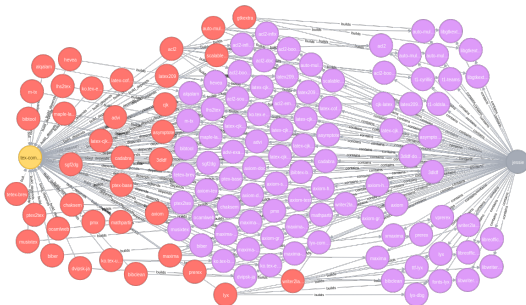
```
match (BP:bp)<-[:build_depends]-(VSP:vsp)-[:builds]->  
  (VBP:vbp)<-[:contains]-(S:suite)  
  where BP.name="tex-common" and S.name="jessie"  
  return BP, VSP, VBP, S
```

Checking build-deps



Find all packages in Jessie that build depends on some version of `tex-common`:

```
match (BP:bp)<-[:build_depends]-(VSP:vsp)-[:builds]->  
  (VBP:vbp)<-[:contains]-(S:suite)  
  where BP.name="tex-common" and S.name="jessie"  
  return BP, VSP, VBP, S
```



Most depend on package in sid



Number of packages in sid that build depend on X , ordered by number of depending packages

```
match (S:suite)-[:contains]->(VBP:vbp)-[:builds]-  
      (VSP:vsp)-[:build_depends]-(X:bp)  
where S.name = "sid"  
with X.name as pkg, count(VSP) as cntr  
      return pkg, cntr order by -cntr
```

Most depend on package in sid



Number of packages in sid that build depend on X , ordered by number of depending packages

```
match (S:suite)-[:contains]->(VBP:vbp)-[:builds]-  
      (VSP:vsp)-[:build_depends]-(X:bp)  
where S.name = "sid"  
with X.name as pkg, count(VSP) as cnt  
return pkg, cnt order by -cnt
```

gives: debhelper: 55438, dh-python: 9289, pkg-config: 9102



Conclusions

Lessons learned



- ▶ Finding a good representation is tricky – see below for future work
- ▶ Don't use Cypher for importing any reasonable amount of data
- ▶ Conversion from an old/grown RDB is a pain
- ▶ Starting from scratch for a new application is fun
- ▶ Visualization in Chrome/Firefox is often a pain – depending on version and OS either the one or the other is better (why? no idea!)

Future work – time allowing



- ▶ Include the bug database
- ▶ Include also intermediate releases by parsing the UDD table for uploads
- ▶ Rework dependency management
I don't like the current status: I would prefer if the dependency points into the tree of vbp and has only an attribute for the relation type.
- ▶ After all that, rewrite the UDD dashboard and see how far it simplifies the SQL code.
- ▶ More graph theoretic: find dependency cycles, connected components etc



Sources for the scripts as well as the slides are available on the Github project:

<https://github.com/norbusan/debian-graph>



Sources for the scripts as well as the slides are available on the Github project:

`https://github.com/norbusan/debian-graph`

Thanks for the attention