Adventures in Extension Packaging

David Wheeler
PGXN, Tembo

PGConf.dev
2025-05-14

HELLO!

I'M DAVID WHEELER

Principle Architect, Tembo

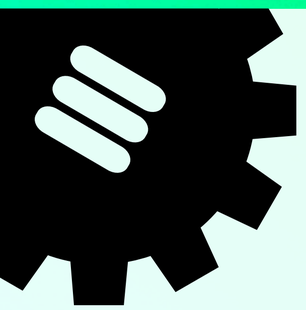
PostgreSQL user since 2000

PostgreSQL contributor since 2008

pgTAP: Unit testing for PostgreSQL

Sqitch: Database change management

PGXN: PostgreSQL Extension network



PGXN

pgxn.org

Established 2011

Source code distribution network

PostgreSQL extensions and tools

Search, documentation, tags

CLI by Daniele Verazzo

PGXN

PostgreSQL Extension Network

users tags recent

in Distributions PGXN Search

acl administration aggregate aggregate function
analytics arithmetic array arrays audit auditing
compatibility count custom background worker data type
datatype dictionary distinct estimate example
extension external data fdw
foreign data wrapper

PGXN, the PostgreSQL Extension network, is a central distribution system for open-source PostgreSQL extension libraries.

Recent Releases

pg_task 2.1.16
PostgreSQL and Greenplum job scheduler pg_task allows to execute any sql command at any specific time at background asynchronously

pg_mustach 1.0.16
PostgreSQL mustach

uint128 1.0.1

EXTENSIONS

422 Extensions

398 Distributions

2490 Releases

468 Users

2 Mirrors

But...

joelonsql/PostgreSQL-EXTENSIONs.md: 1,190

It’s not enough.

🚀 THE PGXN v2 VISION

PGXN V2 FEATURES

Canonical source registry

Search & discover all extensions

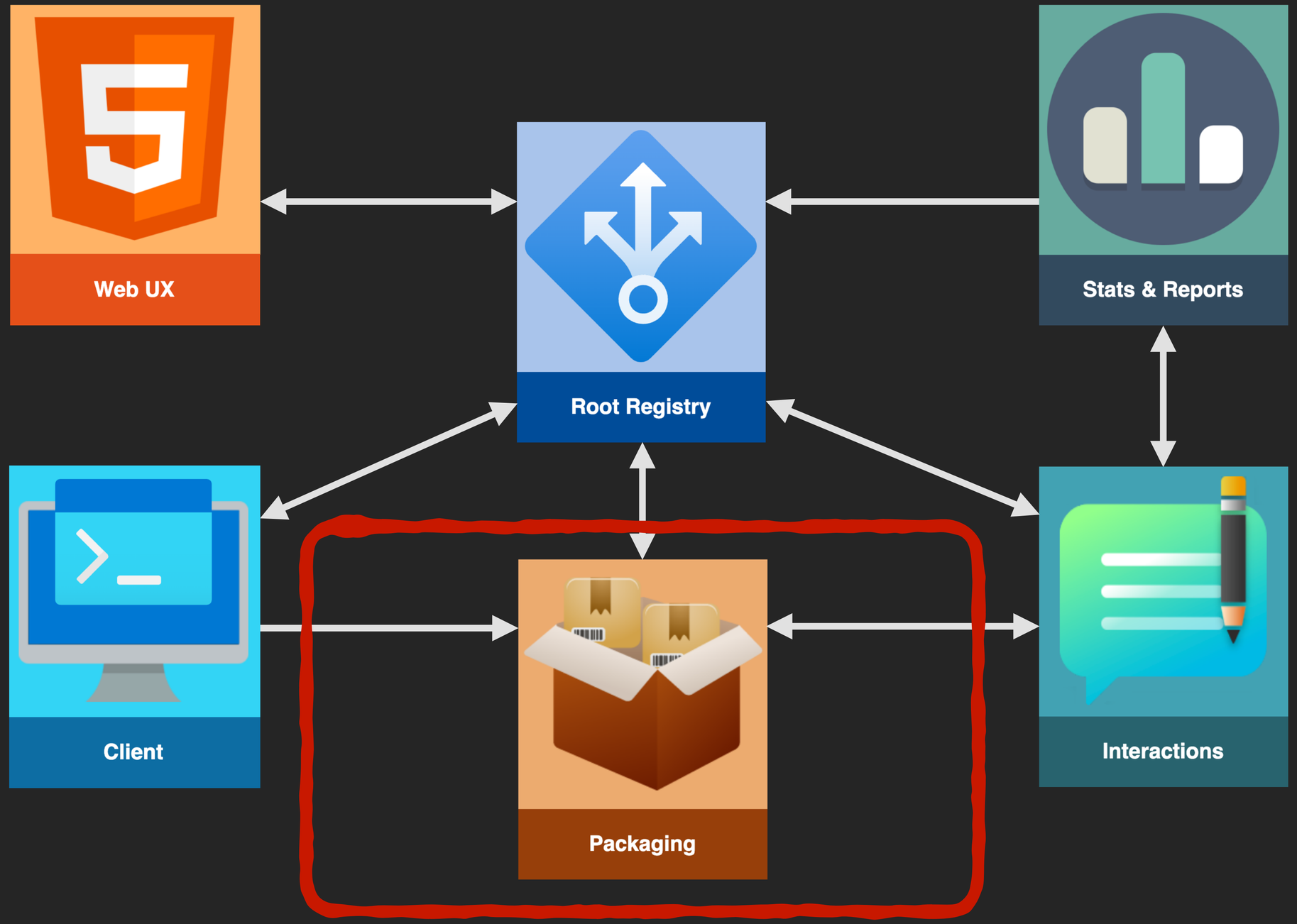
Streaming & federation

Quality metrics & reports

Binary packaging & distribution

- Multi-platform & multi-version

Simple installation & package management


Web UX
Root Registry
Stats & Reports
Client
Packaging
Interactions

METADATA & PACKAGING RFCS

github.com/pgxn/rfcs/pulls

Started developing RFCs

- Meta Spec v2
- JWS Release Signing
- Binary Distribution Format

Feedback appreciated!

Goals:

- META.json-driven packaging
- OCI distribution

META SPEC v2

github.com/pgxn/rfcs/pulls

Indexing: Contents

- Extensions, modules and apps

Discovery: Classifications

- Tags and categories

Dependencies:

- Postgres, build tooling, other extensions, external

```
{
  "dependencies": {
    "postgres": {
      "version": ">= 12.0, < 17.0",
      "with": [ "xml", "uuid", "perl" ]
    },
    "pipeline": "pgrx",
    "packages": {
      "configure": {
        "requires": {
          "pkg:cargo/cargo-pgrx": "==0.14.2",
          "pkg:generic/cmake": 0
        }
      },
      "test": {
        "requires": {
          "pkg:postgres/pg_regress": 0,
          "pkg:pgxn/pgtap": "1.1.0"
        }
      },
      "run": {
        "requires": {
          "pkg:pgxn/hostname": 0
        }
      }
```

Also: build, develop

purl: Package URL

```
{
  "variations": [
    {
      "where": {
        "platforms": ["linux"]
      },
      "dependencies": {
        "packages": {
          "run": {
            "recommends": {
              "pkg:pypi/auto-gptq": 0,
              "pkg:pypi/xformers": 0
            }
          }
        }
      }
    }
  ]
}
```

IMAGINE THIS

Source code release on PGXN

Interactions service calls registry Webhooks

    Yum, APT, Homebrew, Pigsty, DBDev, etc.

Registries automatically pull new release

Install/configure dependencies from spec

Build, test, package, release

Report results back to PGXN

CLI to install packages for each platform

Package Management

```
❯ pgxn search json

  rating  |          package           | version |    date    |
----------+----------------------------+---------+------------+----------
 ⭐⭐      | hadi/json_fdw              | 1.0.0   | 2013-11-18 | Foreign Dat
 ⭐⭐⭐⭐  | theory/jsonschema          | 0.1.0   | 2024-04-30 | JSON Schema
 ⭐⭐⭐    | jma/yamltodb               | 0.10.0  | 2022-11-08 | Database ch
 ⭐⭐      | shanekilkelly/bedquilt     | 2.1.0   | 2016-09-12 | A json docu
 ⭐⭐⭐    | furstenheim/is_jsonb_valid | 0.0.1   | 2017-08-17 | JSONB valid
 ⭐⭐⭐    | postgrespro/jsonb_schema   | 1.0.0   | 2020-08-24 | Decouple js
 ⭐⭐      | kungfury/jsoncdc           | 0.1.0   | 2017-11-23 | Translates

❯
```

Package Management

```
❯ pgxn install -v theory/jsonschema

Installing theory/jsonschema 0.1.0 into pgenv/16.3…
Found theory-jsonschema-0.1.1+2.pg16+1.darwin-arm64.zip
Unpacking theory-jsonschema-0.1.1+2.pg16+1.darwin-arm64.zip
Installing lib/jsonschema.dylib in ~/.pgenv/pgsql-16.3/lib/
Installing share/jsonschema—0.1.0sql in ~/.pgenv/pgsql-16.3/share/extensio
Installing share/jsonschema.control in ~/.pgenv/pgsql-16.3/share/extension
Installing doc/jsonschema.md in ~/.pgenv/pgsql-16.3/share/doc/

Done!
❯
```

Package Management

```
❯ pgxn package ls
Packages installed in pgenv/16.3:

      package       | installed | latest |                        description
--------------------+-----------+--------+-----------------------------------------
 core/citext        | 1.6.0     | ✅     | data type for case-insensitive ch
 core/jsonb_plperl  | 1.0.0     | ✅     | transform between jsonb and plper
 core/unaccent      | 1.1.0     | ✅     | text search dictionary that remov
 theory/jsonschema  | 0.1.0     | ✅     | JSON Schema validation functions

❯
```

📦 UNIVERSAL PACKAGING & DISTRIBUTION

TRUNK PACKAGE FORMAT

File layout defined by the package format

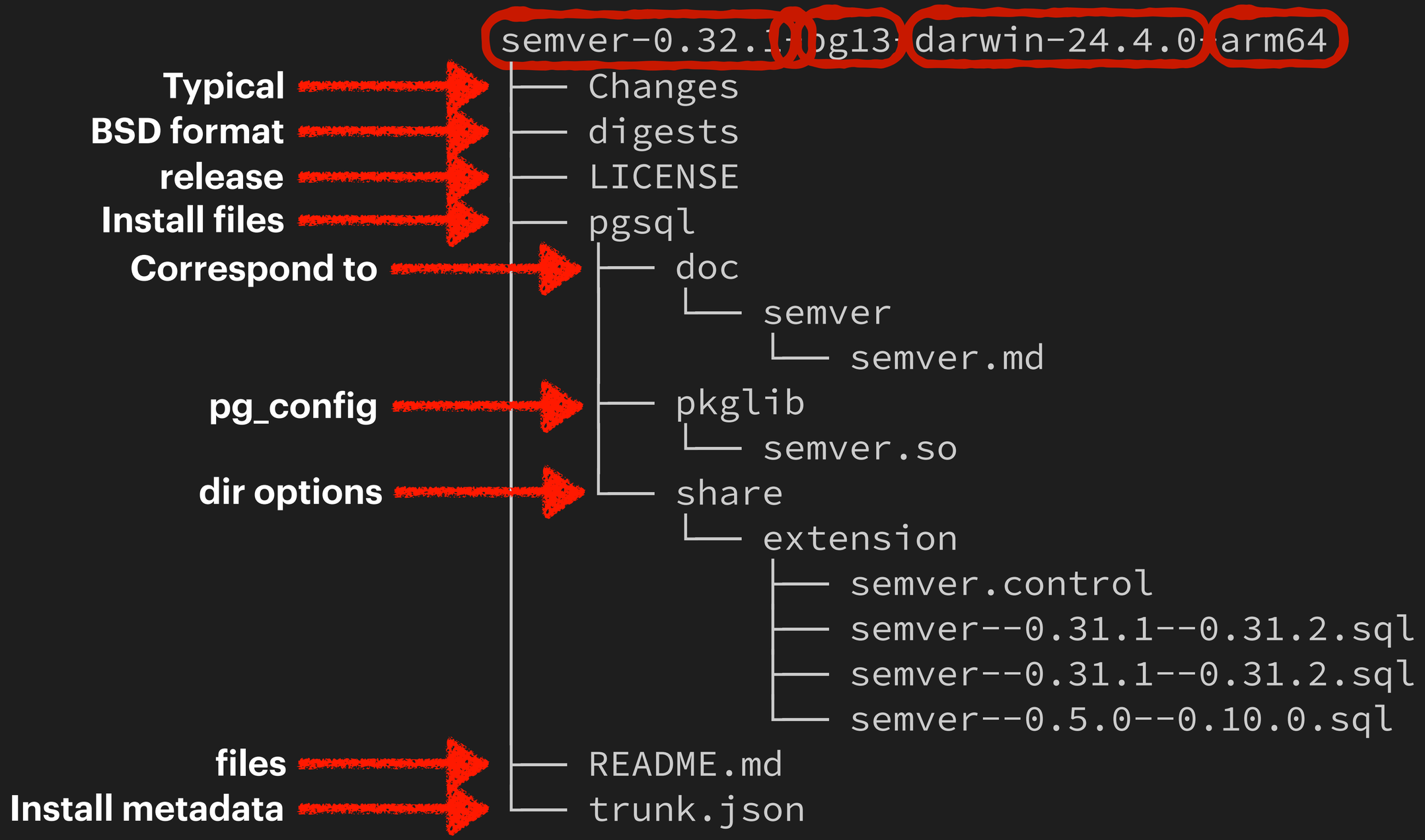
- Not tied to OS layout

Differs from APT/RPM

- Use absolute paths

Allows installation on any compatible Postgres

Regardless of Postgres install location


semver-0.32.1-pg13-darwin-24.4.0-arm64
Typical
BSD format
release
Install files
Correspond to
pg_config
dir options
files
Install metadata
Changes
digests
LICENSE
pgsql
doc
semver
semver.md
pkglib
semver.so
share
extension
semver.control
semver--0.31.1--0.31.2.sql
semver--0.31.1--0.31.2.sql
semver--0.5.0--0.10.0.sql
README.md
trunk.json

trunk.json


```
{
  "trunk": "0.1.0",
  "package": {
    "name": "semver",
    "version": "0.32.1",
    "language": "c",
    "license": "PostgreSQL"
  },
  "postgres": {
    "version": "17.4",
    "major": "17",
    "number": 170004,
    "libs": "-lpgcommon -lpgport -lxml2 -lssl -lcrypto -lz -lreadline -lm ",
    "cppflags": "-I. -I./ -I/Users/david/.pgenv/pgsql-17.4/include/server -I/Users/david/.pgenv/
    "cflags": "-Wall -Wmissing-prototypes -Wpointer-arith -Wdeclaration-after-statement -Werror=v
    "ldflags": "-L/Users/david/.pgenv/pgsql-17.4/lib -isysroot /Library/Developer/CommandLineTool
  },
  "platform": {
    "os": "darwin",
    "version": "24.4.0",
    "arch": "arm64"
  }
}
```

OCI DISTRIBUTION

Compatible with OCI artifact format

Just bundle files into an image

Index platforms in image index manifest

- Automatically install correct binary

Can extend metadata for Postgres version

- Plus any other platform metadata

Standard OCI Metadata


```
{
  "org.opencontainers.image.created": "2024-06-20T18:07:24Z",
  "org.opencontainers.image.licenses": "PostgreSQL",
  "org.opencontainers.image.title": "semver",
  "org.opencontainers.image.description": "A Postgres data type Semantic Versions.",
  "org.opencontainers.image.source": "https://github.com/theory/pg-semver",
  "org.opencontainers.image.vendor": "PGXN",
  "org.opencontainers.image.ref.name": "0.32.1",
  "org.opencontainers.image.version": "0.32.1",
  "org.opencontainers.image.url": "https://github.com/theory/pg-semver"
}
```

Standard Platform Metadata

```
{
  "os": "darwin",
  "os.version": "23.5.0",
  "architecture": "arm64",
  "created": "2025-05-14T18:07:24Z"
}

{
  "os": "linux",
  "architecture": "amd64",
  "created": "2025-05-14T18:07:24Z"
}
```

Custom PGXN Metadata

```
{
  "org.pgxn.trunk.pg.version": "17.4",
  "org.pgxn.trunk.pg.major": "17",
  "org.pgxn.trunk.pg.version_num": "170004",
  "org.pgxn.trunk.version": "0.1.0"
}
```

🔗 DEMO

🤔 THAT'S THE DREAM

Trunk

Download Contribute Docs Blog

Join Trunk on Slack

A Postgres

Extension Registry

Get Trunk

Cargo `cargo install pg-trunk` Copy

Trunk is an open-source package installer and registry for PostgreSQL extensions.

Featured 7

Analytics 22

Auditing / Logging 8

All Extensions

Search 238 extensions

Search

PACKAGING AUTOMATION CHALLENGES

Custom configuration (e.g., Rust package, RUSTFLAGS)

Binary compatibility (e.g., libc)

Packaging variation (e.g., Yum vs. APT)

Dependency package availability

Conflicting dependencies (e.g, pg_duckdb vs duckdb_fdw)

Specific pgrx versions

Patches

Inconsistent build pipelines

Separate releases for different Postgres versions (pgaudit, pg_hint_plan)

Extra build steps (bson, timescaledb)

NOT INSURMOUNTABLE

Add additional build configuration

Balance against exploits

- No commands!

Merge additional metadata

Auto-report failures upstream

Build binaries for each libc

Build binaries for each platform

Start focused, grow from there

Risk of combinatorial explosion?

How I imagine Christoph and Devrim right about now...


AND THEN HE SAID...
WE'LL PACKAGE EVERY EXTENSION
FOR EVERY PLATFORM!

Or maybe...


AND THEN HE SAID...
UPSTREAM MAINTAINERS WILL FIX BUILD FAILURES!

⛅ A CNPG SIDE QUEST

EXTENSIONS IN CNPG

CloudNativePG containers immutable

Extensions must be baked in

Postgres 18 adds `extension_control_path`

Kubernetes 1.33 adds image volumes (beta)

CNPG 1.26 adds extensions to immutable containers

CNPG With Extension

```yaml
apiVersion: postgresql.cnpg.io/v1
kind: Cluster
metadata:
  name: postgresql-with-extensions
spec:
  instances: 1
  imageName: ghcr.io/cloudnative-pg/postgresql-trunk:18-devel
  postgresql:
    extensions:
      - name: pgvector
        image:
          reference: ghcr.io/cloudnative-pg/pgvector-18-testing:latest
  storage:
    storageClass: standard
    size: 1Gi
```

ImageVolume resource

CNPG EXTENSION DEPLOYMENT

1. Triggers rolling update, replicas first
2. Each extension mounted as read-only volume

   `/extensions/pgvector`

3. CNPG Updates GUCs

```
extension_control_path = '$system:/extensions/pgvector/share'
dynamic_library_path   = '$libdir:/extensions/pgvector/lib'
```

CNPG CHALLENGES

This works!

Some limitations

- Volume resolves at pod startup
- Updating GUCs requires restart
- Cannot use single prefix for all image volumes
  - Each operates like full server prefix

Multiple Extensions

```yaml
extensions:
  - name: pgvector
    image:
      reference: ghcr.io/cloudnative-pg/pgvector-18-testing:latest
  - name: semver
    image:
      reference: ghcr.io/example/semver:0.40.0
  - name: auto_explain
    image:
      reference: ghcr.io/example/auto_explain:18
  - name: bloom
    image:
      reference: ghcr.io/example/bloom:18
  - name: postgis
    image:
      reference: ghcr.io/example/postgis:18
```

Multiple Extensions

```
extension_control_path = '$system:/extensions/pgvector/share:/
extensions/semver/share:/extensions/auto_explain/share:/extensions/
bloom/share:/extensions/postgis/share'

dynamic_library_path   = '$libdir:/extensions/pgvector/lib:/
extensions/semver/lib:/extensions/auto_explain/lib:/extensions/bloom/
lib:/extensions/postgis/lib'
```

THE IDEAL

Add extensions by patching manifest

No rolling restarts

- ✅ Requires `ImageVolume` without pod start

No GUCs updated

- Except `shared_preload_libraries`
- ✅ Requires different file layout
  - Would also simplify non-CNPG installs

👀 RFC: EXTENSION PACKAGING & LOOKUP

IMAGINE THIS

Single search path GUC

- No magic “postgresql” suffix

Each extension in eponymous directory

- Maybe versioned?

Pre-defined file subdirectory names

Extension Search Path

```
extension_search_path =
  '$system:/extensions:/usr/local/extensions'
```

Extension Directory Layout

```
❯ ls -1 extensions
auto_explain
bloom
postgis
semver
vector
❯ tree extensions/semver
extensions/semver
├── doc
│   └── semver.md
├── lib
│   └── semver.so
├── semver.control
└── sql
    ├── semver--0.31.0--0.31.1.sql
    ├── semver--0.31.1--0.31.2.sql
    ├── semver--0.31.2--0.32.0.sql
    └── semver--0.5.0--0.10.0.sql
```

Postgres looks for directory name

Also: bin, man, html, include, locale

TRUNK FORMAT

Make trunk format identical to extension directory layout

Installation becomes simple:

- Pull package from OCI registry
- Validate signature & contents
- Move entire directory to extension search path

CNPG AND TRUNK FORMAT

Identical pattern for CNPG

- Mount ImageVolume inside `extension_search_path`
- You’re done!*

*Modulo `shared_preload_libraries`, ImageVolume on pod start


FUN WITH DEPENDENCIES

DEPENDENCIES

Many extensions depend on shared libraries

Packagers manage them via normal means

Can Trunk format genericize?

Two approaches:

1. List dependencies, CLI installs from OS packaging
2. Bundle dependencies

Latter required for ImageVolume immutability

PACKAGED DEPENDENCIES

Package dependencies loaded by OS

`ldconfig` caches installed shared libraries

Perfect for packaged solutions

Infeasible for immutable systems

- Cannot install in immutable directory
- Can run `ldconfig` for new library in mutable directory
- Cannot write to cache in immutable directory

BUNDLING DEPENDENCIES

Install dependent shared library with extension module

Compile Postgres with RUNPATH to that directory

No `ldconfig` caching required

`RUNPATH` supported by most OSes

lekensteyn.nl/rpath.html

RUNPATH

```
❯ chrpath $(which postgres)
/usr/lib/postgresql/bin/postgres:
RUNPATH=/usr/lib/postgresql/lib:/var/lib/postgresql/data/mod
```

```
❯ ls -1 /var/lib/postgresql/data/mod
liburiparser.so.1
liburiparser.so.1.0.30
uri.so
```

RUNPATH

```
❯ chrpath /var/lib/postgresql/data/mod/uri.so
/var/lib/postgresql/data/mod/uri.so:
RUNPATH=/usr/lib/postgresql/lib:/var/lib/postgresql/data/mod
```

```
❯ ldd /var/lib/postgresql/data/mod/uri.so
 linux-vdso.so.1
 liburiparser.so.1 => /var/lib/postgresql/data/mod/liburiparser.so.1
 libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6
 /lib/ld-linux-aarch64.so.1
```

Loading with RUNPATH

```
postgres=# show dynamic_library_path;
         dynamic_library_path
-------------------------------------
 $libdir /var/lib/postgresql/data/mod

postgres=# create extension uri;
CREATE EXTENSION

postgres=# select 'https://example.com/'::uri;
         uri
----------------------
 https://example.com/
```

`$ORIGIN`

Use `rpath=$ORIGIN`

Points to same directory as library

Supported by all major Unixes

- `@loader_path` on Darwin (macOS)
- Not supported on Windows

$ORIGIN

```
❯ chrpath --replace '$ORIGIN' /var/lib/postgresql/data/mod/uri.so
/var/lib/postgresql/data/mod/uri.so: new RUNPATH: $ORIGIN

❯ psql -c "select 'https://example.com/'::uri;"
         uri
---------------------
 https://example.com/

❯ chrpath --replace 'NOTHING' /var/lib/postgresql/data/mod/uri.so
/var/lib/postgresql/data/mod/uri.so: new RUNPATH: NOTHING

❯ psql -c "select 'https://example.com/'::uri;"
ERROR:  could not load library "/var/lib/postgresql/data/mod/uri.so":
liburiparser.so.1: cannot open shared object file:
No such file or directory
```

BUNDLING DEPENDENCIES

Extract dependencies from OS packages

Put them in the Trunk `lib` directory

Compile extension with `$ORIGIN`

```
make CFLAGS='-Wl,-rpath,\$$ORIGIN'
```

Postgres itself doesn't even need `rpath`

Success?

Descendant Dependencies

```
❯ chrpath /var/lib/postgresql/data/mod/http.so
http.so: RUNPATH=$ORIGIN:/usr/lib/postgresql/lib

❯ ldd /var/lib/postgresql/data/mod/http.so
 linux-vdso.so.1
 libcurl.so.4 => not found
 libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6

❯ scp dev:libcurl.so.4 /var/lib/postgresql/data/mod/
```

Descendant Dependencies

```
❯ ldd /var/lib/postgresql/data/mod/http.so
 linux-vdso.so.1
 libcurl.so.4 => /var/lib/postgresql/data/mod/libcurl.so.4
 libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6
 /lib/ld-linux-aarch64.so.1
 libnghttp2.so.14 => not found
 libidn2.so.0 => /lib/aarch64-linux-gnu/libidn2.so.0
 librtmp.so.1 => not found
 libssh.so.4 => not found
 libpsl.so.5 => not found
 libssl.so.3 => /lib/aarch64-linux-gnu/libssl.so.3
 libcrypto.so.3 => /lib/aarch64-linux-gnu/libcrypto.so.3
 libgssapi_krb5.so.2 => /lib/aarch64-linux-gnu/libgssapi_krb5.so.2
 libldap.so.2 => not found
 liblber.so.2 => not found
 libzstd.so.1 => /lib/aarch64-linux-gnu/libzstd.so.1
 libbrotlidec.so.1 => not found
```

Descendant Dependencies

```
❯ scp dev: libnghttp2.so.14 /var/lib/postgresql/data/mod/
❯ ldd /var/lib/postgresql/data/mod/http.so
 linux-vdso.so.1
 libcurl.so.4 => /var/lib/postgresql/data/mod/libcurl.so.4
 libc.so.6 => /lib/aarch64-linux-gnu/libc.so.6
 /lib/ld-linux-aarch64.so.1
 libnghttp2.so.14 => not found
 libidn2.so.0 => /lib/aarch64-linux-gnu/libidn2.so.0
 librtmp.so.1 => not found
 libssh.so.4 => not found
 libpsl.so.5 => not found
 libssl.so.3 => /lib/aarch64-linux-gnu/libssl.so.3
 libcrypto.so.3 => /lib/aarch64-linux-gnu/libcrypto.so.3
 libgssapi_krb5.so.2 => /lib/aarch64-linux-gnu/libgssapi_krb5.so.2
 libldap.so.2 => not found
 liblber.so.2 => not found
 libzstd.so.1 => /lib/aarch64-linux-gnu/libzstd.so.1
```

WORKAROUND?

Works only for direct dependencies

Needs more experimentation

Maybe: Compile dependencies with own RUNPATHs

- Cannot copy from OS packages
- Unless they start using RUNPATH

Meanwhile, only bundle direct dependencies

🤔 FUTURE WORK

PROJECT STATUS

- The vision remains in tact
  - Accessible, easy-install extensions everywhere
- Close to completing v1 build SDK
  - Supports PGXS, pgrx
  - First deliverable: CLI
- Work will inform RFCs
- Update RFCs, discover patterns

STUFF I'D LIKE TO DO

Finish working out Trunk format and dependency patterns

Develop and submit `extension_search_path` patch

Submit ImageVolume feedback to Kubernetes

- Allow runtime mounting!

Start building and distributing OCI Trunk packages

- Make pattern available for distributed registries
- Make your own Trunk releases!

Hack fully-dynamic extension loading into CloudNativePG

LET'S TALK!

Thank You

Adventures in Extension Packaging
David Wheeler
PGXN, Tembo

PGConf.dev
2025–05–14