THE FUTURE OF THE POSTGRES EXTENSION ECOSYSTEM

DAVID E. WHEELER
PGXN, TEMBO

BEGINNINGS

LEGACY

Long history of extensibility

Two approaches

- shared_preload_libraries
- Pure SQL (including PLs)

A few intrepid adopters

- PostGIS
- BioPostgres
- PL/R
- PL/Proxy
- pgTAP

- Postgres has a long history of extensibility
- In the old days, there were two basic approaches to extending Postgres without forking it
    - 1. Loading dynamic shared objects in shared preload libraries
    - 2. Pure SQL extensions that used procedural languages and SQL features to create objects
- There were quite a few intrepid adopters in those years
    - Including PostGIS, BioPostgres, PL/R, PL/Proxy, pgTAP, and others

POSTGRES 9.1

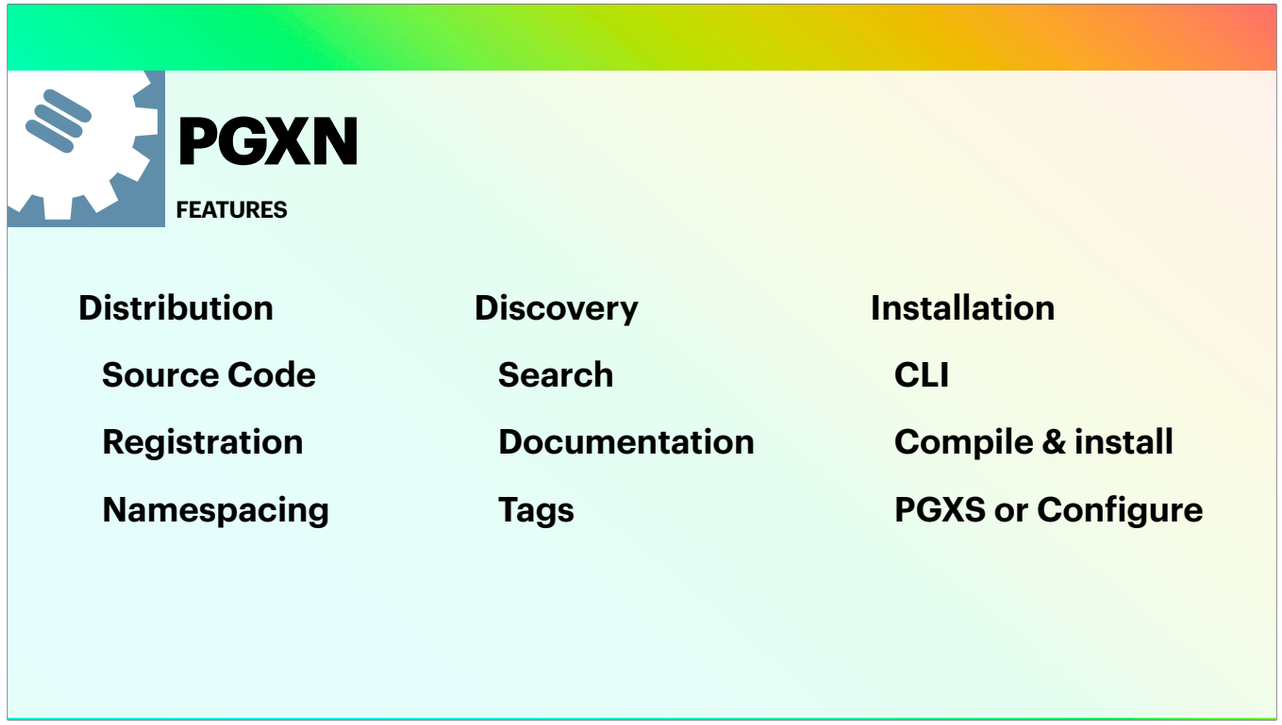
Dimitri Fontaine adds extensions

Key features:

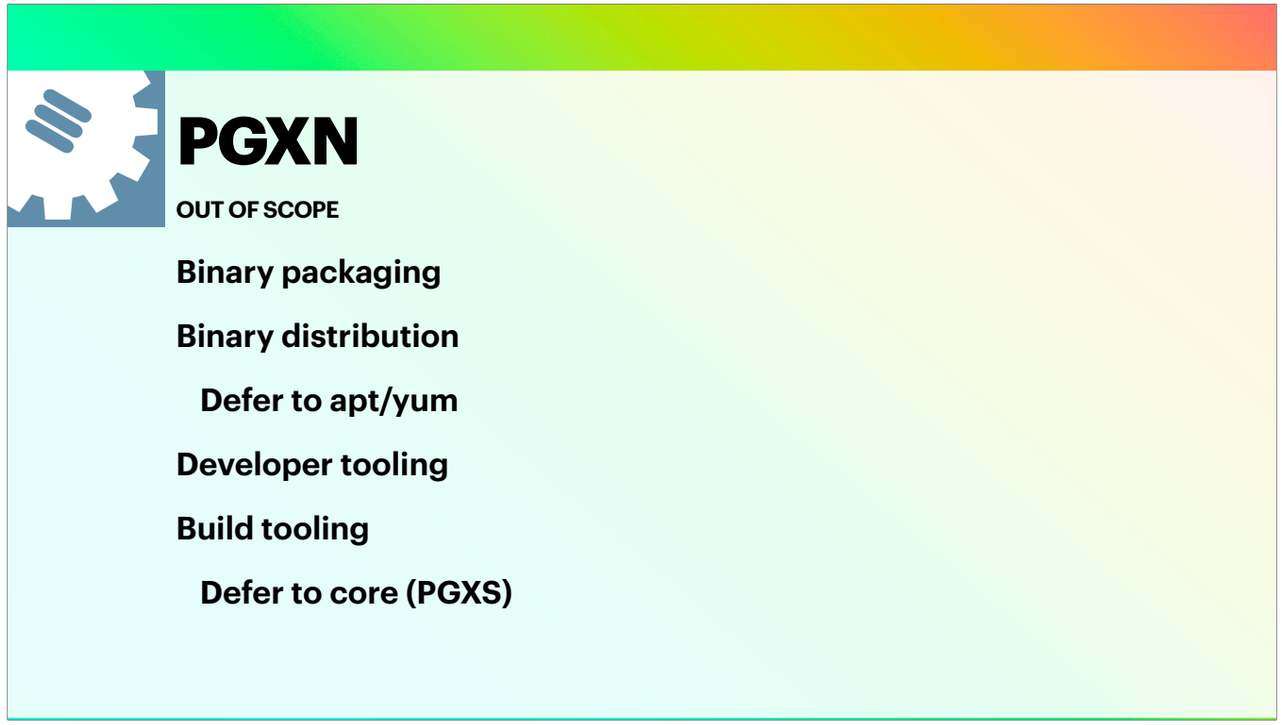
- **Compile and install**
- **CREATE/UPDATE/DROP EXTENSION**
- **pg_dump and pg_restore**

- Then Dimitri Fontaine submitted a patch for formal extension support, which was committed and released in Postgres 9.1 in 2011
- This work built on those existing patterns for extending Postgres, but formalized things through three key features:
  - Tooling to compile and install extensions, both DSOs and pure SQL
  - New SQL commands to create, update, and drop extensions, which bundle together objects into named units
  - And backup and restore support to ensure consistent versioning and behavior when upgrading Postgres itself

I was pretty excited at the possibilities around that time, publicly claiming that “PostgreSQL today is not merely a database, it’s an application development platform.”



- As a result, I proposed to create PGXN, a service for distributing Postgres extensions, with the goal to be the canonical source for all publicly-available extensions
- The features were to include distribution of source code, as well as user registration and namespace management, so there would be no conflict between named extensions
- PGXN also aimed to provide comprehensive search features, and the ability to browse and read well-written documentation. Individual releases could also be tagged for better discoverability.
- A command-line client, meanwhile, would provide a simple interface to download, compile, and install extensions.



- Now, a number of features were out-of-scope of that vision
- These include binary packaging; PGXN would be for source code distribution only
    - It seemed too much to take on support for a wide variety of platforms, so we deferred binary distribution to the community Apt and Yum repositories
- PGXN also would not include developer tooling
- Or Build tooling. There were already solutions for building extensions
    - Most notably PGXS and whatever future directions the Core project might take to support extension authors

- I launched the project in 2010, around the time Dimitri started developing formal extension support
- By December, the little fundraiser I set up had met its goal and it was time to get to work
- The site, pgxn.org, launched in April of 2011 with the first few extensions and a public REST API
- Meanwhile, Daniele Varrazzo took it upon himself to write the command-line client. It was on my list but he beat me to it! Which was cool, because he wrote it in Python, deftly demonstrating that any language could use the APIs and interfaces
- Almost to prove the point, Dickson Guedes released a suite of dev utils written in Ruby in June 2011. The community involvement was so great!

• In other words, I would say, “In classic SDLC fashion, the PGXN POC shipped as an MVP and was neglected.”

⏳ TIME PASSED...

UPTAKE

PGXN: 396 Extensions/369 Distributions

joelonsql/PostgreSQL-EXTENSIONs.md: 1,186

CPAN: 217,860 Perl modules/45,137 distributions

PyPI: 543,205 Python projects

RubyGems: 180,839 Ruby Gems

crates.io: 146,401 Rust Crates

- What has adoption looked like? As of this spring,
- Joel on SQL has inventoried almost 1200 publicly-available extensions in a wide variety of locations, especially GitHub
- And PGXN has 369 distributions with 396 extensions
- Meanwhile, CPAN has 45K distributions with 217K modules
- PyPI has over half a million Python projects
- RubyGems has 180K gems
- And crates.io 146K Rust crates

- This leads me to wonder: why has uptake been so…modest?

LOST OPPORTUNITIES

POSTGRES EXTENSIONS:

Difficult to find and discover

Under-documented and difficult to understand

Maturity difficult to gauge

Hard to configure and install

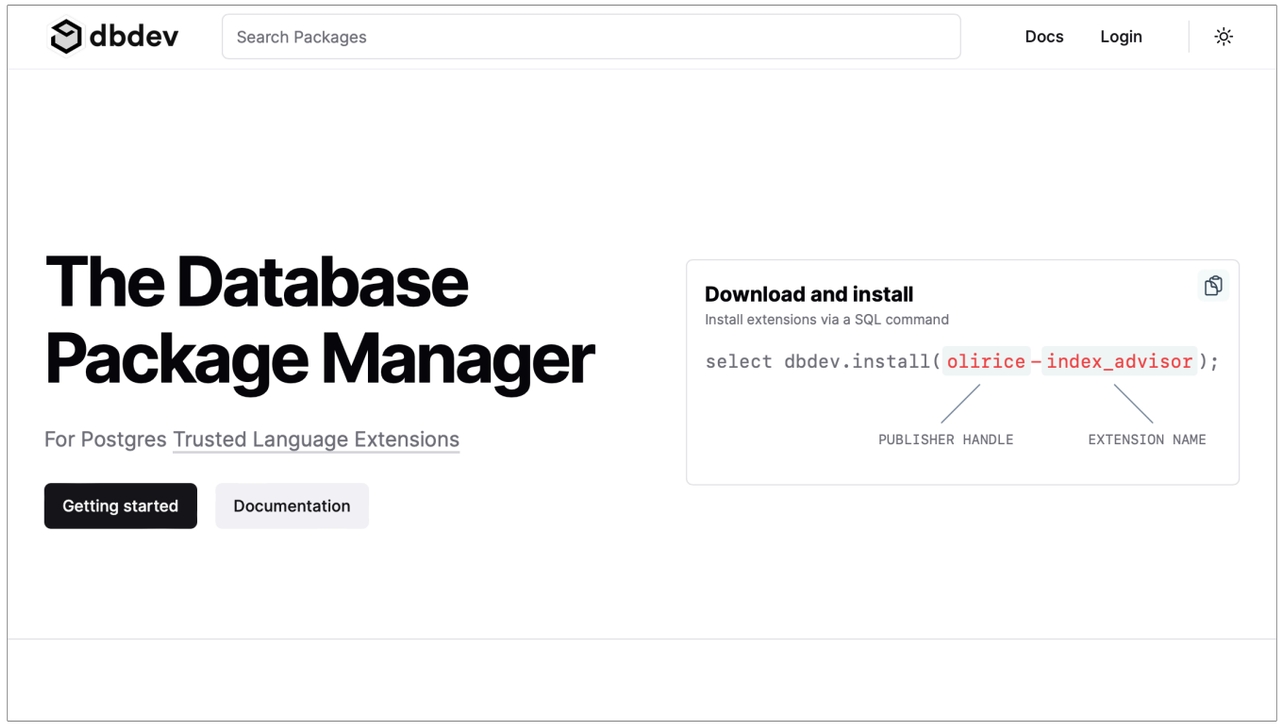
No comprehensive binary packaging

Centralized source distribution insufficient

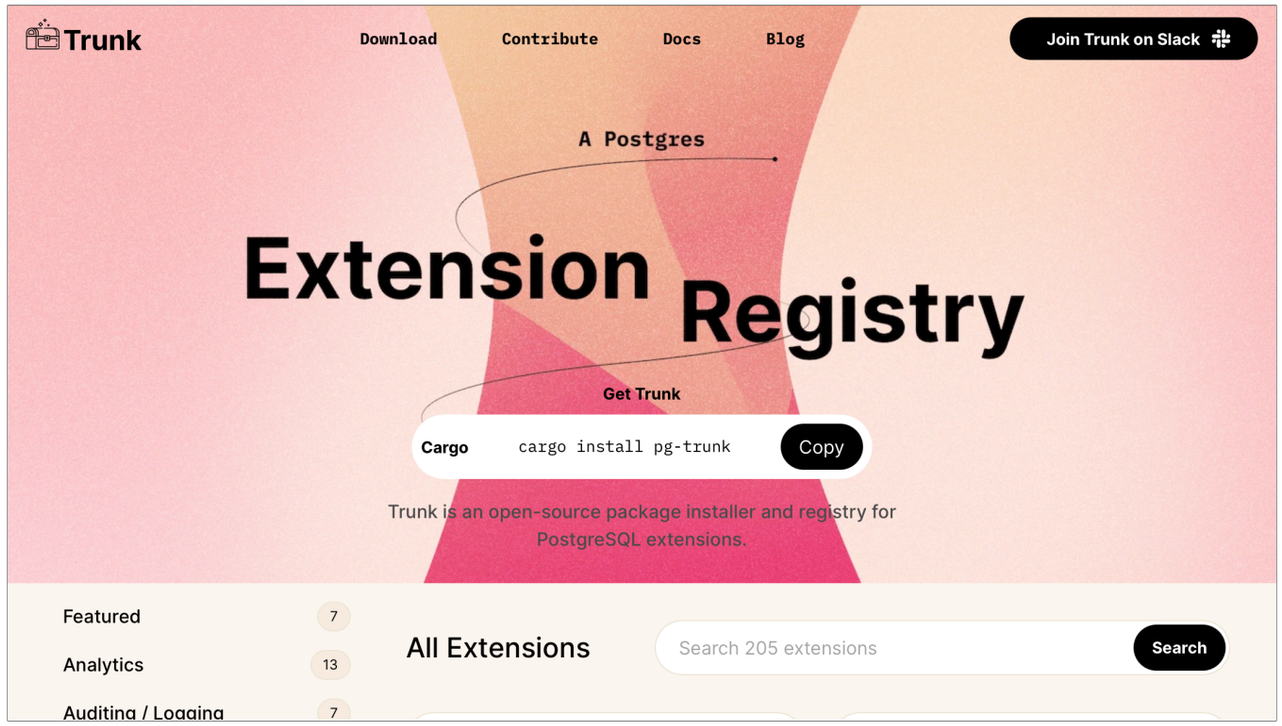
Insufficient developer tooling

- Despite the best of intentions, some opportunities were never met.
- Postgres extensions remain quite difficult to find and discover, since they’re scattered to the internet winds
- Most that one can find, including on PGXN, are under-documented and difficult to understand
- When you do find them, it can be tricky to gauge the maturity, reliability, and stability of an extension, especially if there isn’t much documentation
- Furthermore, they’re difficult to configure and install. Most users just want the add them to their dev and production clusters, and not have to install a bunch tooling like compilers and devel packages on those servers
- This is because there is no comprehensive binary packaging system covering a wide variety of platforms, architectures, and Postgres versions. It’s catch-as-catch-can between the community Apt and Yum repositories and tools like Homebrew. Windows is pretty much omitted altogether.
- Clearly the centralized distribution of source code is insufficient to meet the needs of people who just want to quickly find, install and use extensions.
- Part of the problem is insufficient developer tooling. One has to suss out how to create and maintain extensions by following scattered blogs and example repositories, and the lack of features like documentation standards increase the difficulty of providing consistent product

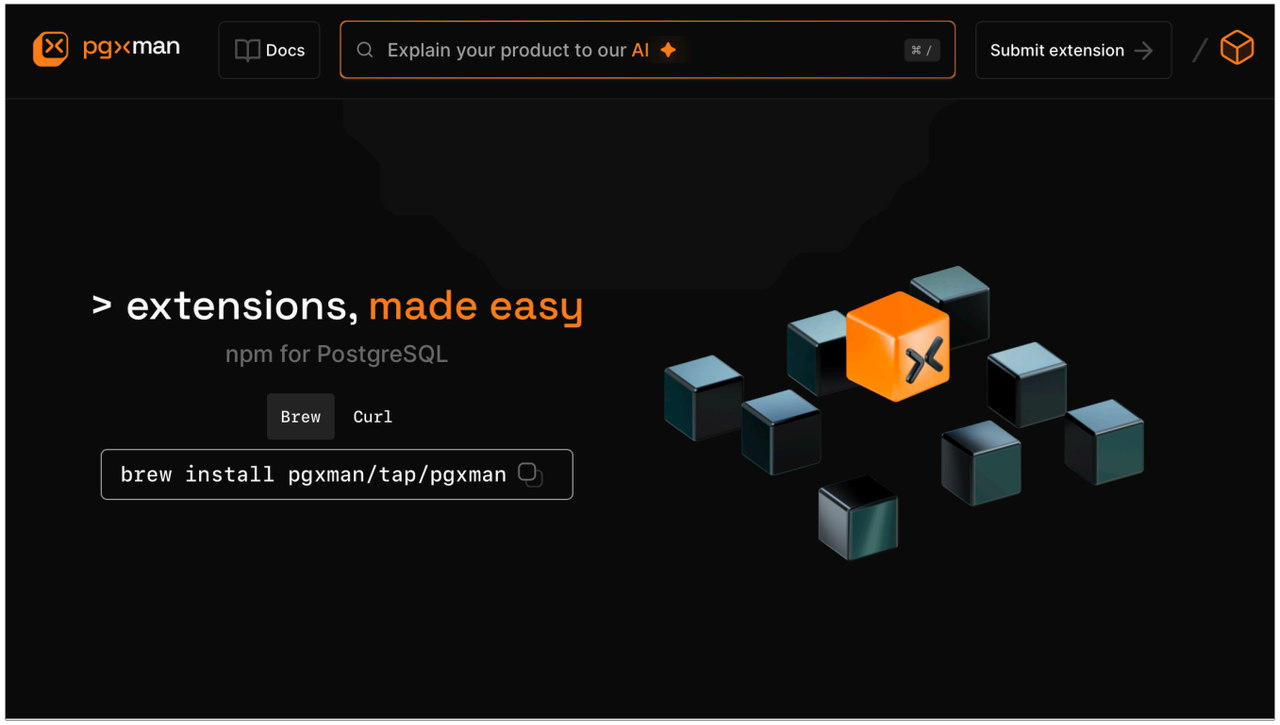
But as I said, things haven’t stood still. A number of new projects have emerged in the last couple years that attempt to fill some of the development and distribution gaps.



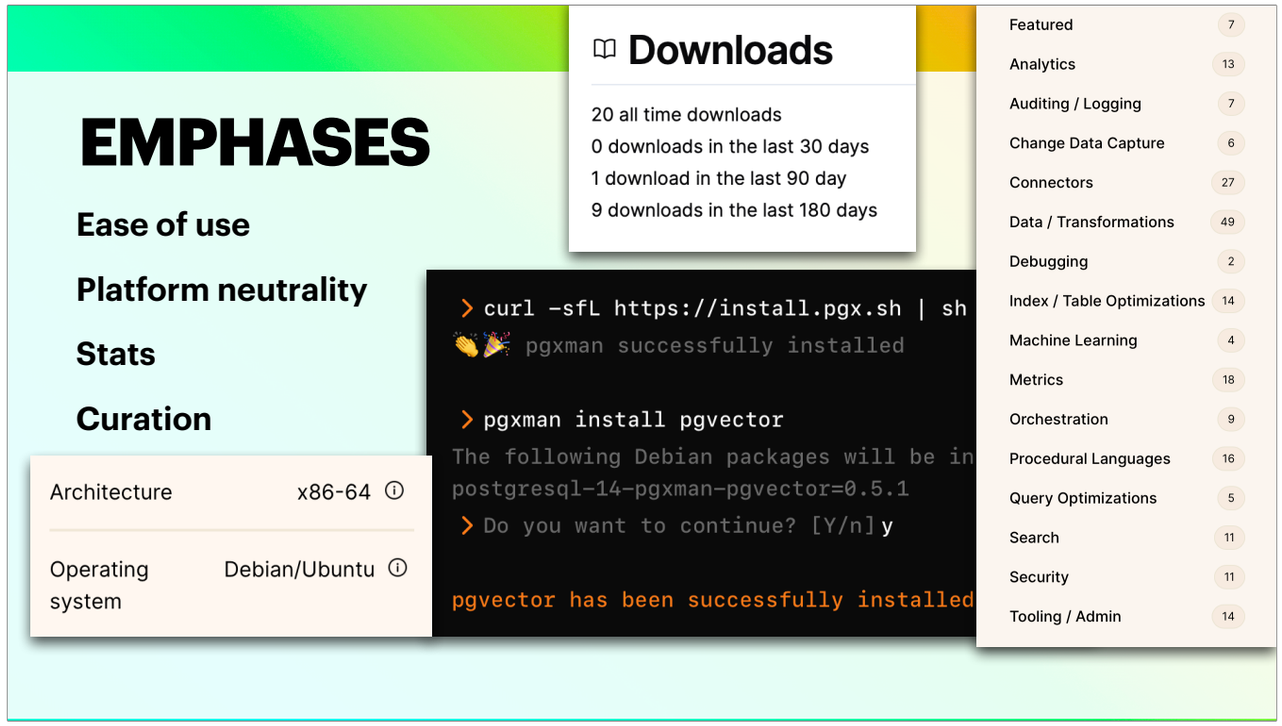
database.dev, or “DB Dev”, is a new packaging registry for trusted language extensions. This service makes it easy to download and install TLEs right in the database!



- Tembo, meanwhile, developed Trunk, hosted at pgt.dev, which provides binary packaging for over 200 extensions. Today it supports only AMD64 Linux, but plans to support a wide variety of platforms and OSes in the future.



Similarly, PGXMan from Hydra provides Debian packages for many extensions, and plans to support for macOS packages.



- These binary registries have emphasized features that PGXN has not. These include
- Ease of use, as in this demonstration of installing PGXMan and a first extension in just two commands.
- Platform neutrality, as in Trunk showing the available platforms for an extension
- Statistics, such as DB Dev listing download numbers for an extension, which can be helpful for evaluating maturity and community acceptance
- And curation, as in Trunk's use of categories for extensions, which have proven a popular vector for discovering extensions to address issues in specific problem domains

MVPS

Trunk	PGXMan	dbdev
Pull request integration	Form-based submission	TLEs
Postgres 14–16	Postgres 12–16	CLI publishing
Debian/Ubuntu-only	Debian/Ubuntu-only	Postgres 11-16
AMD64-only	AMD64 & ARM64	No C/pgrx
210 Extensions	Apt-only	pg_tle && pgsql-http
	81 Extensions	Count Unknown

- Despite these new emphases, currently these new packaging services are best considered MVPs
- Trunk accepts new and updated extensions only by pull request, supports Postgres 14–16, runs only on AMD64 Debian/Ubuntu systems, and currently contains 201 extension
- PGXMan accepts new and updated extensions by a web-based form, supports Postgres 12–16 on AMD64 or ARM64 Debian/Ubuntu, only uses Debian packaging, and contains just 81 extensions
- dbdev meanwhile offers only TLEs, no C or pgrx extensions, but does allow publishing by a CLI. It supports Postgres 11–16 and requires the pg_tle and pgsql-http extensions. I could’t determine how many extensions it contains.

- All these recent developments have invigorated the broader extension ecosystem. But many of the promises we saw early on sadly have yet to be fulfilled.

STATE OF AFFAIRS

Despite early promise…

No single source

Many incomplete attempts

Poor discovery

Difficult installation

Insufficient docs

Low adoption

But…

To whit

- There is no single source for a complete list of all extensions.
    - Every attempt and doing so remains incomplete
- This makes discovery difficult, as one has to know to search PGXN, GitHub, GitLab, Bitbucket, the Postgres Wiki, various cloud providers, specific web sites like postgis.org, and more
- Installation is difficult outside the packages provided by your chosen installer (if you can find a list of them). No one wants to compile extensions on their production database server.
- Most extensions that do exist have very little documentation, usually just a README. And the lack of a documentation standard means that docs, when they exist, vary tremendously in quality and format
- All this has led to pretty low adoption rates. Few databases use extensions at all, and the number using more than one is vanishingly small
- Still, as we've seen…

“There sure has been a lot of excitement around extensions lately.”

Which brings us to the future. What new opportunities are there to improve the extensions ecosystem for developers and users?

PGXN V2

The New World Order

Started this year

Made possible by Tembo

Consider emerging patterns

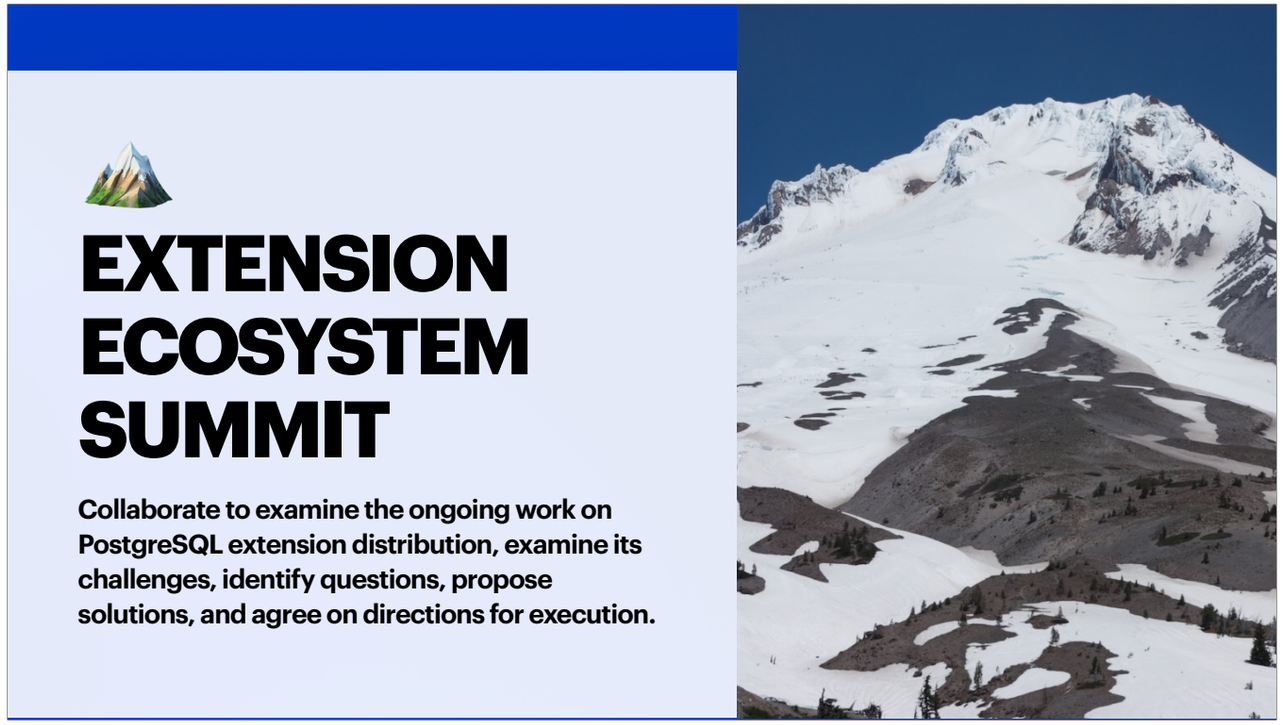
Meditate on deficiencies

Engage deeply with the community

Design and implement new architecture

Serve the ecosystem for next decade

- To answer those questions, we've launched a project, code-named "PGXN v2"
- To find out and make it happen.
- This came about thanks to Tembo, the creators of Trunk, who hired me specifically to
- Consider the emerging patterns, like binary registries and development frameworks
- Meditate on the deficiencies, including discovery and ease of installation
- To engage deeply with the broader community of Postgres developers, extension authors, and users
- In order to understand the use cases well enough to design and implement a new architecture
- That will serve the community for the next decade or more



- One of the key points of community engagement has been the Extension Ecosystem summit. We had six virtual “mini-summits” to share ideas, discover new patterns, and discuss issues, as well as an in-person Summit here on Tuesday. It went great!

WHO IS THE ECOSYSTEM?

First up, let’s talk about the people who create extensions, extension authors.

👨🏽‍💻 EXTENSION AUTHORS

Needs

Tools to create extensions

Learning materials

Standard patterns

Testing tools

Metadata management

Distribution & discovery

Packaging

- Extension authors create and release extensions. They need
- Tools to create and manage extension projects
- Standard patterns for different languages and types of extensions
- Tools for testing on a variety of platforms an Postgres versions
- Tools to manage extension metadata
- Services that distribute extensions broadly and makes them easy to find
- Packaging services to supply binaries for easy installation and adoption of their extensions

👩🏽‍💻 EXTENSION AUTHORS

Delivery

- **Documentation**
  - How-tos
  - Examples
  - Tutorials
  - References
  - Book?
- **CLI for tasks**
  - Metadata
  - Testing
  - Scaffolding
  - Release
  - Documentation
- **CI/CD Automation**
- **Canonical Registry**
  - Discovery
  - Binary Packaging
  - Category Curation
- **Incentives**
  - Quality Metrics
  - Documentation standards
  - Test coverage & matrices

- To get them what they need, we propose to create
- Extensive documentation, including how-tos, examples, tutorials, references, and perhaps a book
- A command-line client to manage metadata, multi-platform and Postgres-version testing, project scaffolding, and documentation
- And pipelines that use that CLI to enable CI/CD automation
- We’ll provide a canonical registry of extensions for discovery, community-provided binary packages, and category curation
- And incentives to help authors improve their extensions, including quality metrics, documentation standards, and published test coverage and pass/fail matrices

Install Management

```
❯ pgxn installs ls

      install      |                        path
-------------------+-------------------------------------------------
 homebrew/14.12    | /opt/homebrew/Cellar/postgresql@14/14.12
 pgenv/16.3        | /Users/david/.pgenv/pgsql-16.3
 pgenv/17devel     | /Users/david/.pgenv/pgsql-17devel
 postgres.app/16.3 | /Applications/Postgres.app/Contents/Versions/16/

❯ pgxn manage pgenv/16.3
Now managing extensions for pgenv/16.3
❯
```

- First, let’s see what Postgres installs we can manage
- This host has four installs: one from Homebrew, two from pgenv, and the Postgres.app
- The “manage list” command also shows the root directory for each installation, like this one for the pgenv-installed Postgres 16.3
- Let’s tell the CLI to manage that install

Develop CLI

```
❯ pgxn develop new --extension --language c --vcs github bike
Scaffolding C extension “bike”…
   ⚙️ Create bike.control
      META.json
      bike.control
      Makefile
      README.md
      src/bike.c
      sql/bike--0.0.1.sql
      doc/bike.md
      test/sql/base.sql
      test/expect/base.out
      .github/workflows/ci.yml
      .github/workflows/release.yml
    ✨ Init repository
       git init bike
```

- We use the pgxn develop new --extension command to scaffold a new extension, including META.json, control, Makefile, and README
- Since we specified the C language, it also scaffolds a C source file as well as the SQL migration script and a documentation file
- It also creates a first pg_regress test
- And GitHub workflows for CI/CD

Develop CLI

```
❯ pgxn develop add-upgrade 0.0.2

Adding upgrade from 0.0.1 to 0.0.2…
   Rename sql/bike--0.0.1.sql to sql/bike--0.0.2.sql
   Create sql/bike--0.0.1--0.0.2.sql
   Update META.json
          bike.control
```

- Now let’s say the developer needs to make a new version. They use the pgxn develop add-upgrade command
- It renames the versioned SQL script and adds a new one for upgrade commands, and records the new version in the META.json and control files

Develop CLI

```
❯ pgxn develop require 14

Require PostgreSQL 14…
   🆙 Update META.json

❯ pgxn develop require semver 0.32.0

Requiring semver 0.32.0…
   🆙 Update META.json
              semver.control

❯
```

- We can use the CLI to manage dependencies, too, like requiring a specific version of Postgres
- Or another extension

Develop CLI

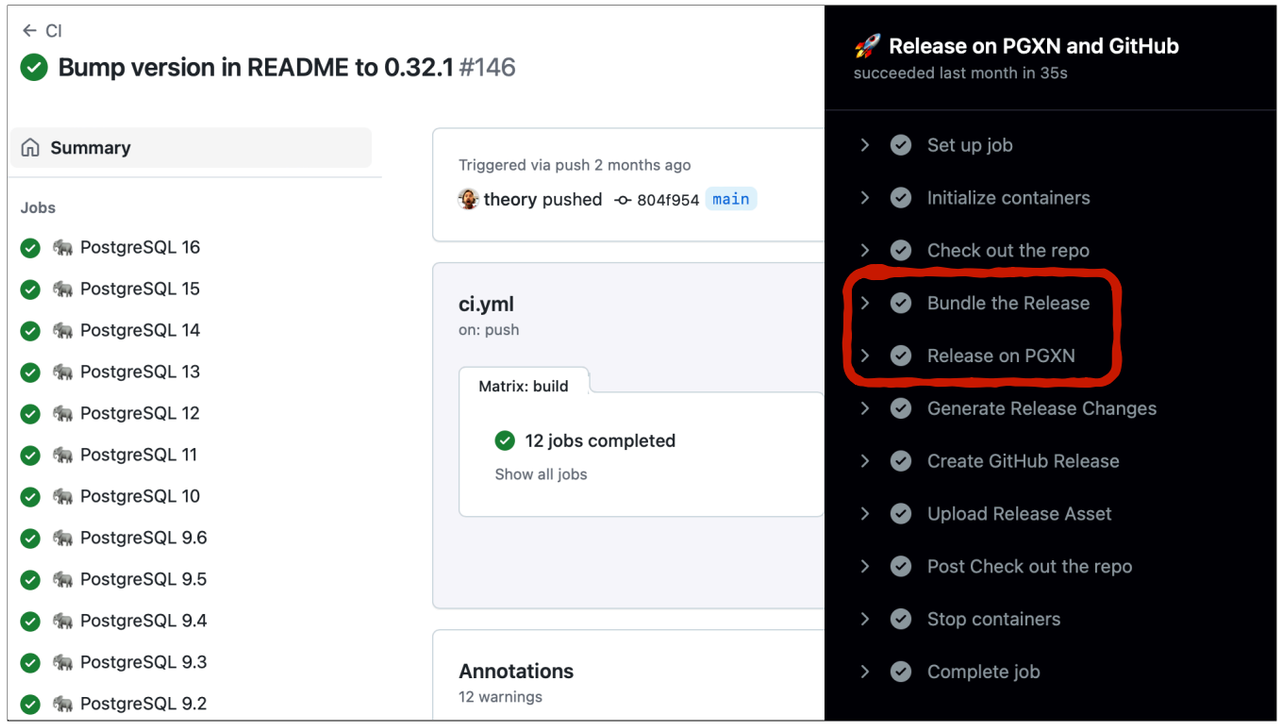
```
❯ pgxn develop test --username postgres

Testing bike 0.0.2 in pgenv/16.3
🔨 make
   gcc -c -o src/pair.o src/pair.c
   gcc src/pair.o -o src/pair.so
📂 make install
   install -c -m 644 bike.control '/pgsql/share/extension/'
   install -c -m 644 sql/bike--0.0.2.sql  '/pgsql/share/bike/'
   install -c -m 644 sql/bike--0.0.1--0.0.2.sql  '/pgsql/share/bike/'
   install -c -m 644 src/bike.so '/pgsql/lib/'
✅ make installcheck PGUSER=postgres
   ok 1         - base                        15 ms
   1..1
   # All 1 tests passed.
```

- The client supports testing, as well.
- It tells us which install it’s testing again
- Compiles the extension, if necessary
- And installs it
- Before running the installcheck command

Develop CLI

```
❯ pgxn develop release

Releasing bike 0.0.2 on PGXN…
    ✅ Validate META.json… Okay
    📦 Bundle bike-0.0.2.zip with git archive…
    🚀 Publish bike-0.0.2.zip on PGXN…

🎉 Done: https://pgxn.org/theory/bike/0.0.2
```

- And of course we can use the CLI to release a new version
- It will validate the metadata, bundle it up for release, and publish it to PGXN.



- The same CLI and commands can be used in CI/CD,
- As in this example that uses it to bundle and release a new version

The next members of the ecosystem we aim to help are the packagers.

📦 EXTENSION PACKAGERS

Needs

Release notification

Build automation

Predictable download URLs

Flexible sourcing

Extensive metadata

Feedback loops

- What do packagers need?
- Notification of new releases, so they know to update packages
- Tools to automate building packages
- Predictable URLs so it’s easy to download extension, given their package names and versions
- Flexible source: some may prefer to download and repackage precompiled packages, while others will need to compile from source
- Extensive metadata will help packagers determine the dependencies needed to build extensions, and which packages will need to depend on at runtime
- Feedback loops, to let authors know when there are issues that need to be addressed.

📦 EXTENSION PACKAGERS

Delivery

Release feeds & notification Webhooks

Package on every release

Build from source or repackage community binaries

Metadata to automate building from source

Report build and test failures

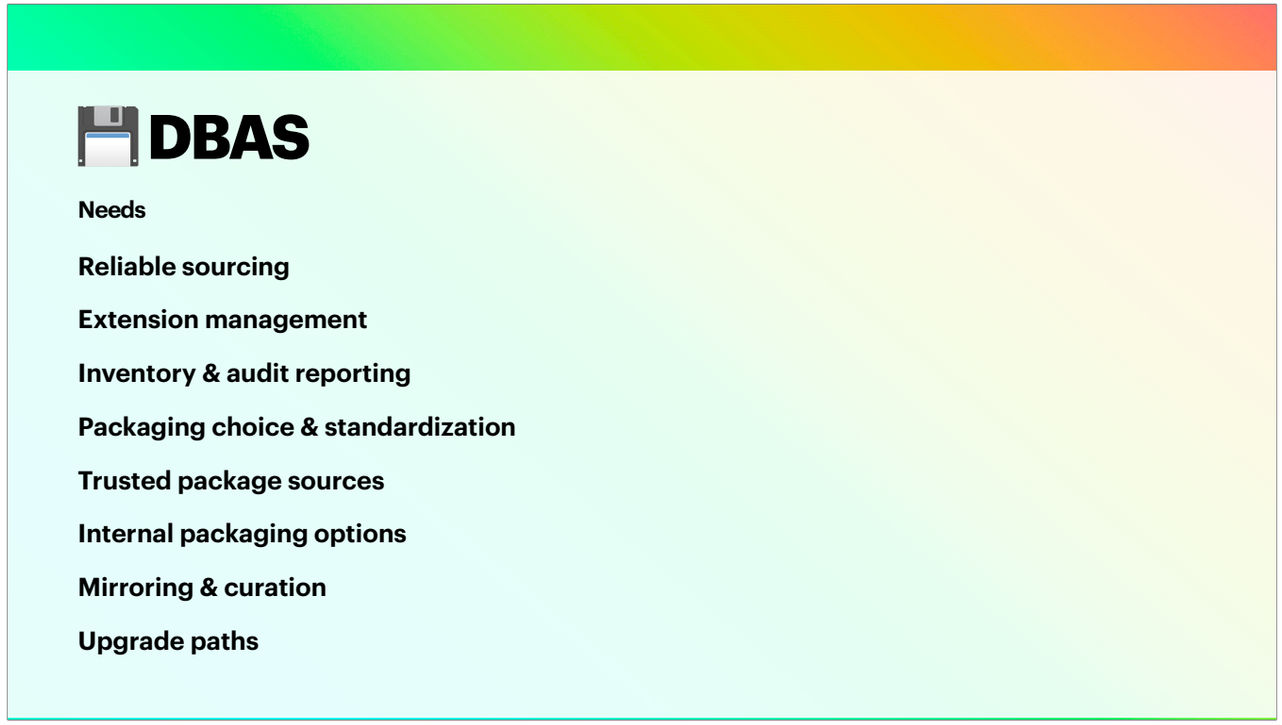
Submit download stats to root registry

- Here’s what we aim to provide to packagers
- Release feeds, like Atom and JSONFeed, and real-time notification webhooks
- These will let them package every release
- And they can build from source or repackage from community binaries
- Expanded metadata will allow packagers to automate packaging by providing the extension and system dependency data required to build and run an extension
- We’ll provide a reporting interface to allow trusted packagers to report back successes and failures that provide quality signal and will help authors identify and fix issues
- They’ll also be able to submit download stats, if they have them, to be aggregated by the root index

```
Name:           bike
Version:        0.0.2
Source0:        https://pgxn.org/dist/theory/bike/bike-%{version}.zip
BuildRoot:      %{_tmppath}/%{name}-%{version}-%{release}-root-%(%{__id_u} -n)

%prep
%setup -q -n bike-%{version}

%build
pgxn package build

%install
pgxn package install --destdir $RPM_BUILD_ROOT
%{_fixperms} $RPM_BUILD_ROOT/*

%check
pgxn package installcheck --username postgres

%clean
rm -rf $RPM_BUILD_ROOT
```

- The CLI will come in handy for packers, as in this RPM spec example
- The source points to a predictable URL
- And the packager can use the CLI's package namespace to build
- Install
- And test an extension

Next up, database administrators.



- What to DBAs need?
- They need reliable sourcing, so they can trust the extensions they instal
- The need extension management, so they can tell what's installed and what's out-of-date
- Inventory and audit reporting, to keep an eye on things and know what's in use and what's not
- Standardized packaging they can trust, and choice
- of trusted sources
- Internal packaging options, for private or in-house-developed extensions
- Mirroring and curation, to control what's available to be installed
- And upgrade paths to ease the path from one extension version to another, and one Postgres version to the next

- To meet the needs of DBAs, we aim to provide
- Extension management tools, including
    - Fast binary installation
    - rather than compiling from source
    - Community binary packages
    - Or community OS packages (yum, apt, etc.)
    - Or, they can build their own packages source
    - The manager will generate reports on installed extensions and their upgrade status
    - Users of an internal registry will have proper access controls
    - And some way to simplify upgrades, perhaps by somehow allowing multiple versions to be installed and used at once
- In addition, we'll build a federated binary packaging and distribution system
    - Which will allow DBAs to run their own registries
    - Where they can package in-house extensions
    - All using the same protocols and tooling as the community registry
    - They can gate access by appropriate authentication or network controls
    - And subscribe to curated, commercial registries for added features like advanced security scanning
- Thanks to the standard protocol, they'll be able to mirror internal registries to upstream registries
- For faster local performance

Package Management

```
❯ pgxn package ls
Packages installed in pgdb/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/semver      | 0.32.0    | 0.32.1 | Semantic version data type

❯
```

- Use the “package list” command to show the extension packages that the CLI manages for that install
- There are three extensions from the Postgres core, plus the semver package from PGXN user theory
- Note the “latest” version listed here. That means there is a newer version available to be installed

Package Management

```
❯ pgxn upgrade -v theory/semver

Upgrading theory/semver 0.32.1 in pgdb-yum/16.3…
Found theory-semver-0.32.1+1.pg16+2.linux-arm64.zip
Unpacking theory-semver-0.32.1+1.pg16+1.linux-arm64.zip
Installing lib/semver.so in /usr/lib/postgresql/16/lib
Installing share/semver–0.32.1.sql in /usr/share/postgresql/16/extensions/
Installing share/semver.control in /usr/share/postgresql/16/extensions/
Installing doc/semver.md in /usr/share/doc/postgresql-doc-16

Done!
❯
```

Also macOS, *BSD, Windows,

- Let’s use the upgrade command to upgrade it!
- Note that it found a binary package for Postgres 16, macOS/Darwin, and the arm64 architecture
- It then installs the DSO in the libdir. No compilation!
- The remaining SQL, control, and documentation files also go where they belong
- And that’s it. We plan to at least support Linux, BSD, Windows, and likely others

Package Management

```
❯ pgxn package ls
Packages installed in pgdb-yum/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/semver      | 0.32.0    | ✅     | Semantic version data type

❯
```

- So what does the list of packages look like now?
- There we go
- Note that semver is now the latest version.

- The availability of packages will be included on the root registry web site, for quick and easy availability vetting by DBAs.

No ecosystem community would be complete without, of course, its users

👤 USERS

Needs

Reliable index of extensions

Effective search

Curation & categorization

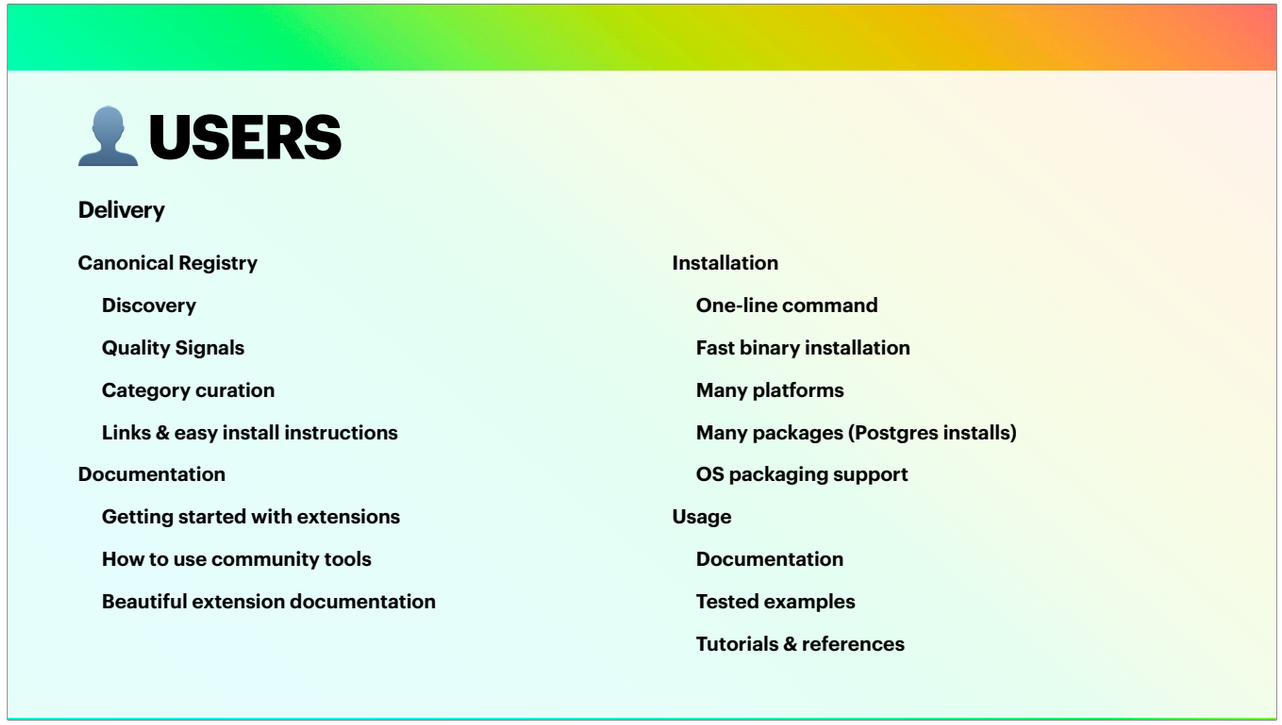
Quality signals

Version support info

Quick & easy installation

Informative documentation

- What do users need?
- A reliable and comprehensive index of extensions, so they can find them
- Effective search, so if they know what sorts of problems to solve they can search for extensions that do so
- Curation and categorization to improve discovery and highlight related extensions often used together
- Quality signals, so they have a sense of how well written and maintained extensions are
- Version support information, to know when an extension is available for their versions of Postgres and OS
- Quick and easy installation, so they can quickly try an extension and get to work
- Informative documentation, to learn how to use extensions



- Here’s what we plan to deliver for users
- The canonical root registry
    - To enable discovery, provide quality signals, curate extensions into categories, and provide links and easy installation instructions
- Documentation
    - To help users get started with extensions
    - Learn to use the community tools
    - And how to use extensions through beautiful documentation
- Installation tooling
    - The ability to install an extension and all dependencies in a single command
    - Very quickly, by providing community-built binary packages
    - With support for a wide array of platforms
    - And Postgres installations
    - Plus, feed into OS packaging systems that users might prefer
- And finally documentation to simplify learning an extension
    - Including tested examples
    - And tutorials and references

Package Management

```
❯ pgxn search json

  rating   |          package           | version |    date    |
-----------+----------------------------+---------+------------+-----------
 ★★        | hadi/ison_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

❯
```

- Lets’s say a user wants to do some JSON validation. We search the registry for “json” to see what’s what.
- It shows the search results, including some stats — in this moc averages from some sort of ratings service
- This one looks decent. It has 4 stars, does JSON schema validation, and has a pretty recent release

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!
❯
```

- Let’s install it.
- As before, the CLI finds a binary package appropriate to this system and Postgres version
- Once again installing the pre-compiled DSO
- As well as the control, SQL, and documentation files

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

❯
```

- So what does the list of packages look like now?
- There we go

And finally, we come to industry, the companies and organizations that participate in the community and provide extension-based products and services.

🏭 INDUSTRY

Needs

Embrace and extend

Extension curation products

Extension rating products

Linked from root registry metadata

Offer extensions for customers

Use APIs to provide UX for DBaaS products

Immutability

- What does industry need?
- The ability to embrace and extend extensions
    - By providing products that curate extensions for customers
    - Or provide extension rating services, and more
- Trusted providers can be linked from the root registry, so users can quickly see what extension are available from their preferred providers
- They can offer extensions for their customers, including proprietary extensions
- And use the community APIs for installation and management, including UX for database-as-a-service products
- And they need immutability, so that their core database offerings can be reliably static even as installed extensions may vary by customer

🏭 INDUSTRY

Delivery

Specialized Registries

Additional metadata, stats, reports, signal

Binary Registry Federation

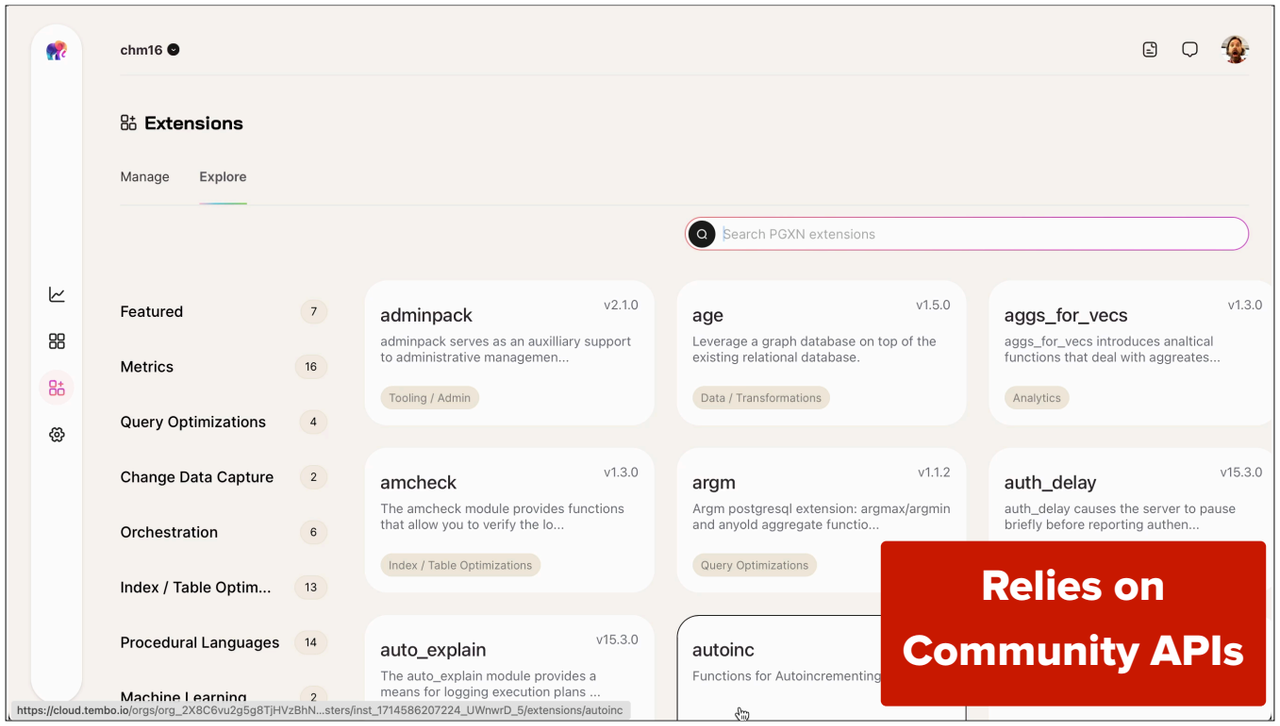
Curate extensions for customers by copying only approved extensions to own registry

Use APIs to provide UX for DBaaS products

Report back stats

Separate mutable extension directory

- Here’s what we plan to deliver for users
- Federated registries, so they can mirror or curate extensions from other registries into their own
- Additional metadata, stats, reports, and other signal that can be contributed back to the root registry, or maintained separately
- Adoption of a federated registry standard so internal registries can use all the same tools and APIs as the community registries
- APIs for installation can be used in a UX just like the community CLI does
- The ability to report back stats to the root registry



- For example, a Postgres as a service provider like Tembo will integrate the API to enable easy installation of binaries via a web UX
- So perhaps we search for semver among all PGXN extensions
- Then, having found it
- Click the install button and get the extension
- This functionality is identical to the CLI installation example, and uses the APIs. All public and private Postgres services will be able to use the service, or to federate and curate extensions for their customers.

🚀 THE VISION

Let’s get to the vision for how we’ll achieve all this

PGXN V2 FEATURES

Canonical source registry

Search & discover all extensions

Streaming & federation

Quality metrics & reports

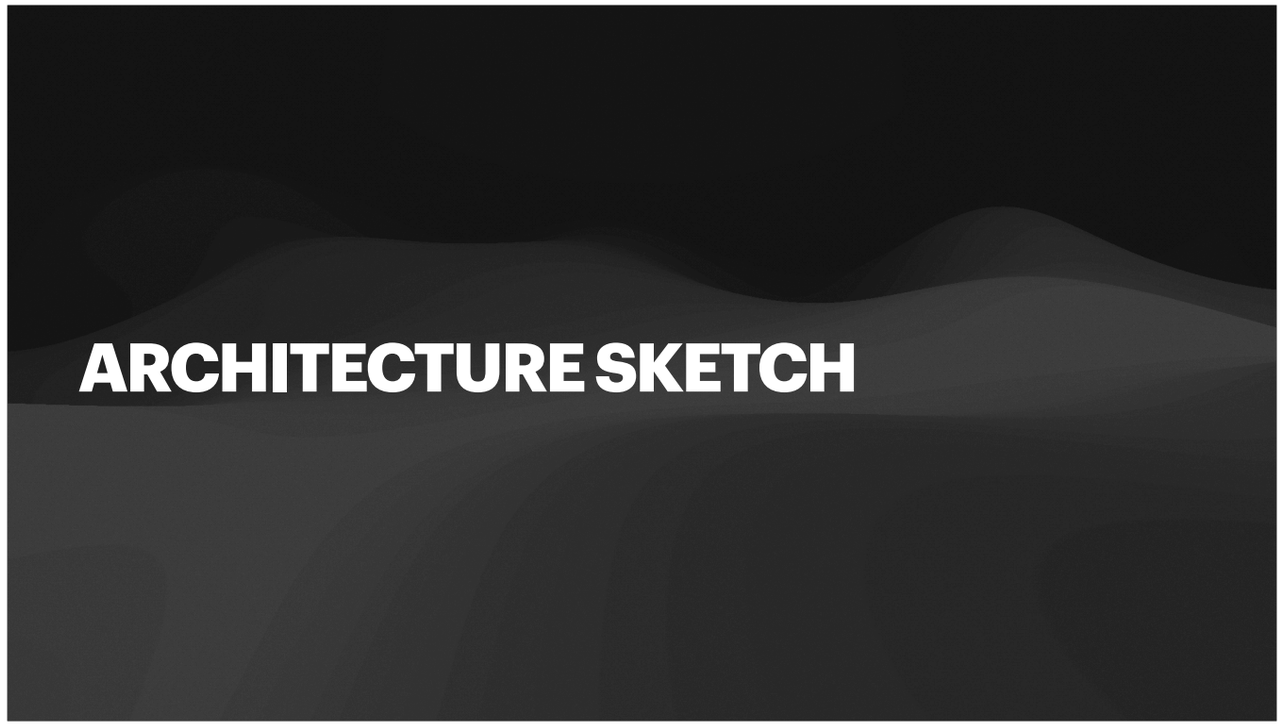
Binary packaging & distribution

Multi-platform & multi-version

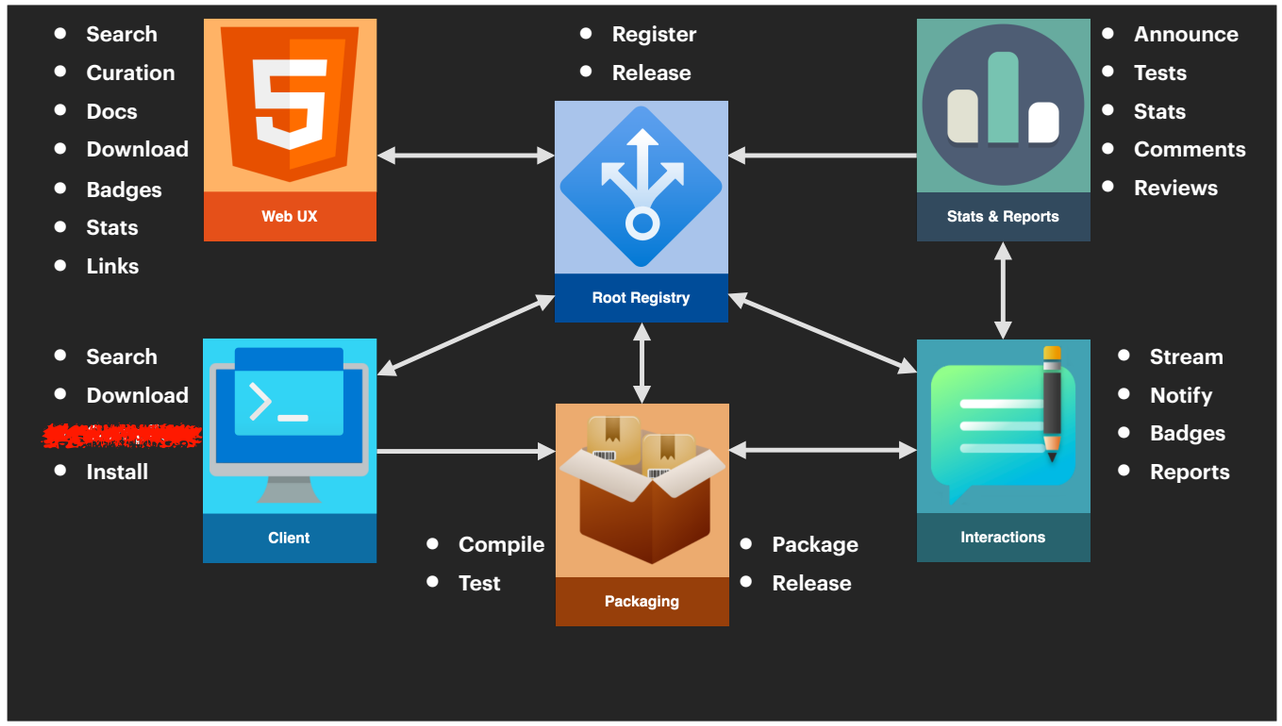
Simple installation & package management

We've sketched some of the features we'd like design and develop toward the ideal extension distribution system of the future, including:

- A canonical source code registry for publicly-available extensions
- APIs and interfaces to search and discover those extensions
- Streaming and/or federation, too enable third parties to build tools and services, such as custom build systems and curated registries
- Quality metrics and reports, to help users evaluate extension quality
- Binary packaging and distribution
    - for a wide variety of OSes, architectures, and Postgres versions
- Simple installation and package management for users to quickly install and start using extensions from the registry



I’ve made a high-level architecture sketch to capture the categories of tools and services to fulfill these jobs.



- At root is, of course, the root registry
- Where authors register accounts and release extension source code
- This is complemented by a well-known web site
- for users to find those extensions, read their docs, look at curation information, and download them
- Next is a command-line client
- which allows one to quickly find, download, compile, and install any of the indexed extensions. These services build on PGXN and other precedents.
- Brand new is the idea of Interactions
- These services stream registry changes like new releases, which subscribers can use for their own purposes. It will also provide WRITE APIs where trusted clients can submit reports, badges, stats, and other metadata about extensions. Think test matrices or Security scores.
- Stats and reports are the categories of services that provide that kind of data, getting notifications from Interactions for recent relases, downloading sources from the root, and submitting results to interactions, which publishes them in the root registry
- From there the Web UX picks them up to show links, badges, and simple stats to help users evaluate extension quality
- Last but not least we have binary packaging services that support a variety of OSes, architectures, and Postgres versions. They subscribe to interactions to learn about new releases
- They pull the source from the root registry
- And use the client to build and package them, then push the data back to the root registry so the appropriate availability data can be visible in the UX.
- The CLI will then also install from the registry, eliminating the need for DBAs to compile from source

JOIN US

Background: https://wiki.postgresql.org/wiki/PGXN_v2

Architecture: https://wiki.postgresql.org/wiki/PGXN_v2/Architecture

Project: https://github.com/orgs/pgxn/projects/1

Tasks: https://github.com/pgxn/planning/issues

Writing: https://justatheory.com/tags/pgxn/

Slack: https://pgtreats.info/slack-invite

- So that’s the vision. We’d love to have you join the effort. Feedback and ideas, in particular, are greatly appreciated.
- The home page for the project is in the Postgres Wiki
- As is a document that goes into much more detail on the proposed high-level architecture
- The project to build all this is managed in a PGXN GitHub project
- As are the tasks to be carried out for each service, tool, and feature
- I’ve also blogged extensively on the project since the beginning of the year at justatheory.com.
- And the best place to stay in the loop and get in involved is the #extensions channel on the community Slack, which you can join here.

THANK YOU

THE FUTURE OF THE POSTGRES EXTENSION ECOSYSTEM

DAVID E. WHEELER
PGXN, TEMBO