# PGConf NYC

Sep 30 - Oct 2, 2024

Microsoft

Follow
Azure Database for PostgreSQL
on LinkedIn

https://aka.ms/azure-pg-linkedin

Microsoft

PGConf NYC

# Meet our Postgres team at PGConf NYC 2024

Microsoft | PGConf NYC

# Extension Development Lifecycle in Rust

Arda Aytekin

Mon Sep 30 | 12:00 EDT

## Agenda

## Background & Context

- AI extensions team at Azure Database for PostgreSQL
- Extension development in Rust
- <u>Our choices</u> and learnings
  - hence, <u>not a definitive set of best practices</u>
- Our constraints
  - Team members having different hardware/OS/setup
  - Conditional enablement of different “features” (e.g., telemetry)
  - Differing build and runtime systems
  - Interfacing with different (low-level) libraries
  - Remote API calls
  - Compliance and security

## Sample Project

- https://github.com/aytekinar/pgconf-nyc-2024
  - A simple vector operations extension
  - Only dot product and vector norm
- Visual Studio Code + Docker + Development Containers
  - Dev container with Rust tooling and PG versions 14, 15 and 16
- Building an extension from scratch in 8 phases/steps

# Why Rust & PGRX

## Why Rust

- Safety and performance
  - Ownership and lifetimes (memory safety)
  - (Zero-cost) High-level abstractions (perf.)
- Toolchain (cargo)
  - Unit tests, doc tests, benchmarks
  - Extensible via custom commands
  - Easy dependency management
- Good resources (even the compiler)
  - Even though the learning curve is steep

## Why PGRX

- Fully-managed development environment
  - create, unit-test, run, install, package
- Target multiple PostgreSQL versions
  - write once, deploy/build everywhere
- Automatic schema generation
- First-class UDF support
- Easy custom types
- Server programming interface
- Executor/planner/(sub)transaction hooks
- Logging through PostgreSQL

## Project Structuring & Cargo Workspaces

- Visual Studio Code + development containers + features

- Files -> Modules -> Crates -> Packages
- Opinionated (but tidy/clean) project structuring

- Cargo workspaces
  - Help manage multiple related packages developed in tandem
  - Same Cargo.lock file and output directory
  - No additional copies of the same dependency downloaded
  - Every crate in every package uses the same version of the same dep.
  - Help save space and ensure compatibility

## Optional Dependencies and Features

- Features provide a mechanism for optional dependencies and conditional compilation
- Optional dependencies are not compiled by default

- cargo-pgrx uses this approach to
  - target/support different PostgreSQL versions (v12…v17)
  - enable the corresponding feature of the dependency
  - support building for and testing against different PG versions

## Testing, Benchmarking and Profiling

- cargo [pgrx] test
  - Unit testing support
  - End-to-end testing support
  - Documentation testing support

- [cargo] criterion
  - Statistics-driven (micro-)benchmarking

- [cargo-]flamegraph and samply
  - Flamegraphing/profiling tools

## Foreign Function Interface

### From C to Rust

- bindgen
  - Automatically generates Rust FFI bindings to C
- cc
  - Library to compile C/C++/assembly/CUDA files into a static archive for Cargo to link into the crate
- cmake
  - Build dependency for running cmake to build native libraries
- libc
  - Necessary definitions for easy C interoperability

### From Rust to C

- cbindgen
  - Creates C/C++ headers for Rust libraries that expose a public C API

# IO-/CPU-Bound Tasks & gRPC

## IO-Bound

- Tokio
  - Asynchronous runtime for the Rust language
  - Single-threaded and multi-threaded runtimes
  - Asynchronous version of the standard library

- Tonic & Prost!
  - Native gRPC client & server implementation with async support
  - Native Protocol Buffers implementation in Rust (Prost!)

## CPU-Bound

- Rayon
  - Data-parallelism library
  - Parallel iterators
  - Expensive CPU-bound operations

- Crossbeam
  - Set of tools for concurrent programming

## Compliance and Lifecycle Management

- cargo pgrx test & cargo pgrx package
- cargo deny
  - **Advisories.** Detect security vulnerabilities and unmaintained crates
  - **Bans.** Denying specific crates and detecting duplicate versions
  - **Licenses.** Verify that each crate has license terms you find acceptable
  - **Sources.** Allow only known/trusted sources and/or vendored file dependencies
- cargo udeps
  - Helps find unused dependencies in Corgo.toml

# Recap

- Rust
  - Safety and performance
  - Extensible package manager (cargo)
  - Tight control via workspaces & features
  - Interoperability with C
  - Compliance & lifecycle management

- PGRX
  - Fully-managed development environment
  - Supports multiple PostgreSQL versions
  - First-class UDF support & custom types
  - Server programming interface
  - Logging through PostgreSQL

## References

### Rust

- The Book
- The Cargo Book
- The Performance Book
- The Rustonomicon

## References

### Frameworks & Tools

- PGRX
- criterion, flamegraph and samply
- Tokio (IO-bound), Rayon (CPU-bound), Crossbeam, and Tonic & Prost!
- bindgen, cbindgen, cc, cmake, and libc
- opentelemetry, opentelemetry_sdk, and opentelemetry-otlp
- cargo-deny and cargo-udeps

Microsoft

Microsoft

## Got 3 minutes? We'd love your input on some of our Postgres work

Get your FREE socks @ Microsoft booth

## Have you listened to?

## TalkingPostgres.com

Save the date
June 10-12, 2025

# POSETTE: An Event for Postgres

2025 Now in its 4th year!

A free & virtual developer event

Subscribe to news → aka.ms/posette-subscribe