Microsoft

# Learnings from Extension Development in Rust & PGRX

Arda Aytekin & Aykut Bozkurt

Apr 26, 2024

# Agenda

1. Background & Context
2. Why Rust & PGRX
3. Project Structuring and Cargo Workspaces
4. Optional Dependencies and Features
5. Foreign Function Interface
6. IO- and CPU-bound Tasks, gRPC Communication
7. Compliance and Lifecycle Management
8. Recap

# Background & Context

- AI extensions team at Azure Database for PostgreSQL
- Two AI-related extensions
  - azure_ai in Nov 2023 (Microsoft Ignite)
  - <another_one> in May 2024 (Microsoft Build)
- Six months of Rust & PGRX efforts
  - Complete remake of a C-based extension
  - Our choices and learnings (not a definitive set of best practices)
- Topics touched
  - Optional dependencies and features (e.g., telemetry)
  - Testing and benchmarking
  - Foreign Function Interface (mostly C)
  - API calls and RPC
  - Compliance

# 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
- 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

- 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 dependency
  - 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
  - Different features for different supported versions (11…16)
  - Enables the corresponding feature of the dependency
  - Supports building for and testing against different PostgreSQL versions from the same codebase

# 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

- Tokio
  - Asynchronous runtime for the Rust language
  - Single-threaded and multi-threaded runtimes
  - Asynchronous version of the standard library
  - IO-bound operations
- Rayon
  - Data-parallelism library
  - Parallel iterators
  - Expensive CPU-bound operations
- Tonic & Prost!
  - Native gRPC client & server implementation with async support
  - Native Protocol Buffers implementation in Rust (Prost!)

# 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

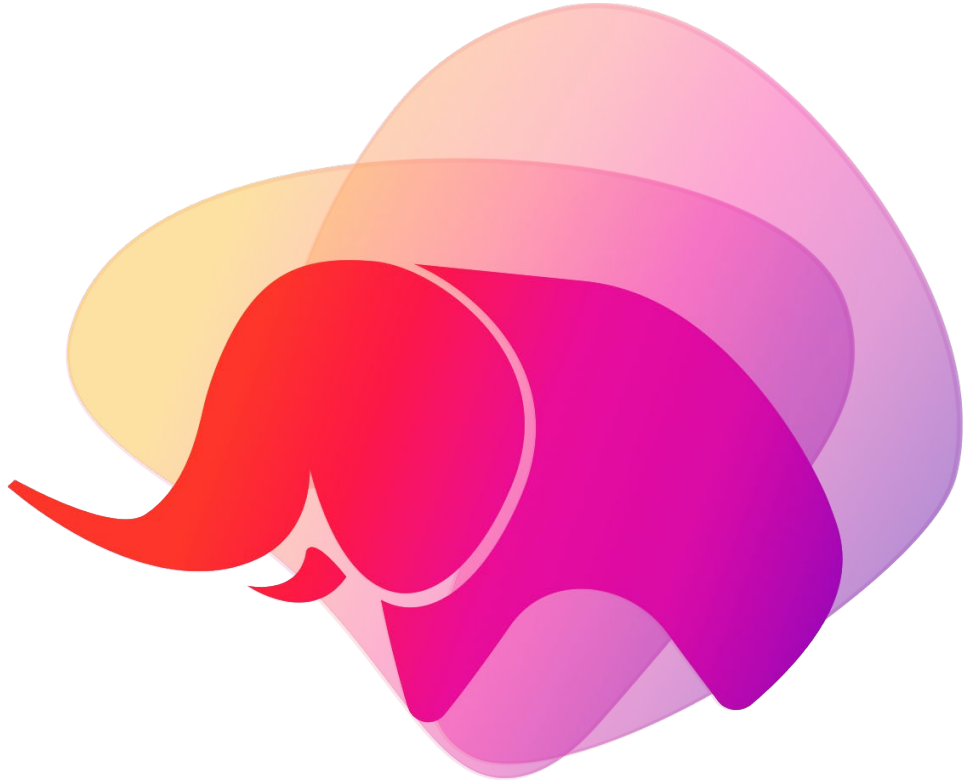
- The Book
- The Cargo Book
- The Rustonomicon

## Frameworks & Tools

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

Save the date
June 11-13, 2024

# POSETTE:
# An Event for Postgres
2024



A free & virtual developer event

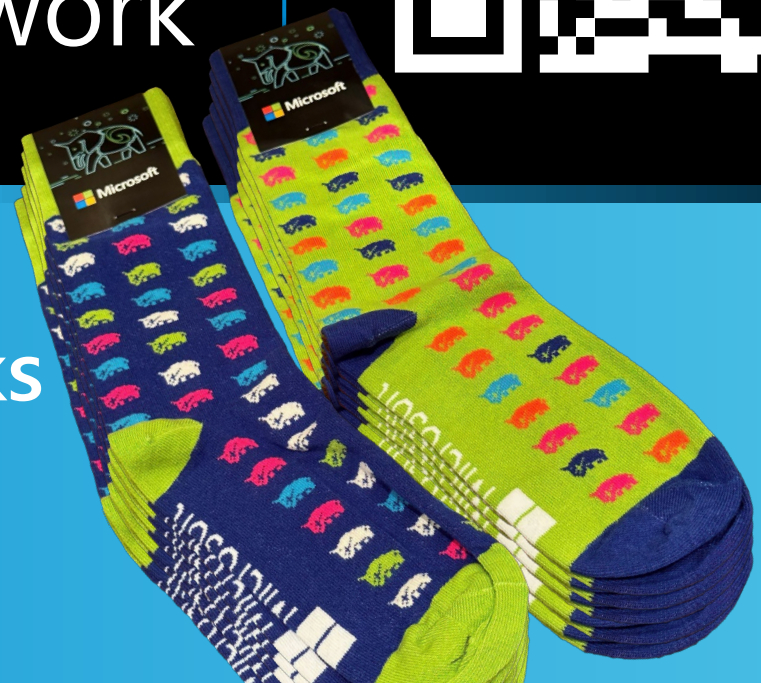
Save the Date → aka.ms/posette-cal

Microsoft

# Got 3 minutes?

# We'd love your input

on some of our Postgres work




Get your FREE socks
@ Microsoft booth