Bazel in polylang repo

protobuf & go & c++
reproducible, incremental and dependency aware builds

Who am I?

- Krzysztof Czapla, Backend engineer
- You can find me on:
  - linkedin
  - github

Goal of the presentation

- demonstrate how bazel can help with common pains in multi language repo
- not a deep dive into bazel and its module internals

Pains before bazel

- each language comes with its own toolchain
- manual protoc
- drift
- generated code committed into the repository
- everything ducked typed with makefile and CI - flakiness

What is bazel?

- build and test tool
- open-sourced by google
- uses high-level, human readable language - starlark (python flavor)
- supports multiple platforms and language

Bazel 101 - source code organization

- source code organized in directory trees called repositories
- workspace is a set of repositories
- source files are organized in nested hierarchy of packages

Bazel 101 - Repository

- directory tree with a boundary marker file at its root
- boundary marker is called MODULE.bazel
- two types of repository
  - main repository - bazel commands run here
  - external repository - defined by repo rules

Bazel 101 - Workspace

- env shared by all bazel commands run from the same main repo
- encompasses the main repo and the set of all defined external repos

Bazel 101 - package

- collection of related files and specification how they can be used to produce output
- package definition - directory that contains BUILD.bazel
  - it includes all files in the directory + directories beneath it without BUILD.bazel file
- package contains targets

Bazel 101 - targets

- files
  - source
    - written by people
  - generated
    - generated from source files
    - not checked in
- rules
  - specify relationship between set of input and set of output files
  - doesn’t matter if input is generated or source file

Bazel 101 - Label

- identifier of the target
  - canonical form: @myrepo//my/app/main:app_binary
    - @myrepo - repository name
    - //my/app/main: - package name
    - app_binary - target name
  - if referring to the target in main repo - //my/app/main:app_binary

Demo

- walk over directory structure
- present bazel files
- build and run
- break build
- show how bazel detects that
- fix
- run

Trade-offs

- It adds new tool on the top of the familiar stack - increases maintenance complexity
- For statical languages, bazel compiles external dependencies
  - fresh build can take long time
  - compilation issues of external libraries
- developer productivity tools support
  - for c++ it is possible to generate compile commands
  - for go, I couldn’t make generated files visible to the gopls

What else can bazel bring to your project?

- extensible - you can write your own rules
- multi platform build support
- caching
- sandboxing
  - each bazel command runs in isolated directory only with inputs defined in the rules and writable output directory - no dependency on global libs and headers, env vars etc.
- patch external libraries on the fly

my experience with polylang repos and bazel

- c++, Makefiles, generated files and java
  - c++ and java generated files distributed as maven package
  - c++ build with makefiles, all duck taped with maven
- c++, protobuf, go and bazel
  - protobufs were compiled into c++ code and linked inside repository
    - better than the first situation
    - however go generated files were generated and committed in the different directory
- Vanilla Envoy
  - Pure bazel
  - go repo used envoy repo and proto package as external repository
  - no files committed
  - Idea for this presentation based on that design

Q&A

The end