Generating bazel definitions

for your nix code

Aleksander Gondek
Artur Stachecki

Bazel eXchange
22nd of June, 2022

Acknowledgements

TWEAG

Visit

Qarik

Visit

What does it do?

```
{ ... }:
stdenv.mkDerivation (finalAttrs: {
  pname = "hello";
  version = "2.12";

  src = fetchurl {
    url = "...";
    sha256 = "1ayhp9v4...";
  };

  buildPhase = ''
    ...
  '';
})
```

$\longrightarrow$

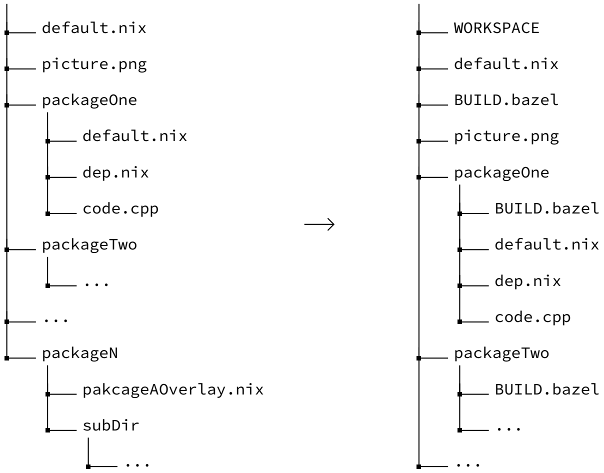
```
http_archive(
    name = "io_tweag_rules_nixpkgs",
    strip_prefix = "rules_nixpkgs-...",
    urls = ["..."],
)

...

nixpkgs_package(
  name = "hello",
  attribute_path = "hello",
  nix_file = "//:default.nix",
  nix_file_deps = [
    "//:nixpkgs.json"
  ],
  repository = "@nixpkgs",
  build_file = "//:hello.BUILD.bazel",
)
```

default.nix

WORKSPACE


default.nix
picture.png
packageOne
default.nix
dep.nix
code.cpp
packageTwo
...
...
packageN
pakcageAOverlay.nix
subDir
...
WORKSPACE
default.nix
BUILD.bazel
picture.png
packageOne
BUILD.bazel
default.nix
dep.nix
code.cpp
packageTwo
BUILD.bazel
...
...

Motivation

Why use nix and Bazel?

- Solves the issues of dealing with ’global dependencies’
- Solves the issue of exponential explosion of container images
- Applies ’bazel mindset’ to global dependencies
- No need to roll out your own self-contain package ’X’

Motivation

Why automate?

- **tweag/rules_nixpkgs**
- Integrate nix derivations and codebase with Bazel
- Remove maintenance and synchronization burden
- Improve approachability of nix + Bazel solutions
- Our inner sloths demanded it

```
$ ./show-and-tell.sh ...
```

Lessons learned

Gazelle

Why?

- **bazelbuild/bazel-gazelle**
- Generator of Bazel-code for go projects
- Various extensions for different languages (python, haskell...)
- Easily extensible via Golang libraries

Gazelle

How?

```
load(
    "@bazel_gazelle//:def.bzl",
    "DEFAULT_LANGUAGES",
    "gazelle_binary",
    "gazelle",
)

gazelle_binary(
    name = "my_gazelle_binary",
    languages = DEFAULT_LANGUAGES + [
        "<bazel-target-golib-extension>",
    ],
    visibility = ["//visibility:public"],
)

load("@bazel_gazelle//:def.bzl", "gazelle")

gazelle(
    name = "gazelle",
    gazelle = "//:my_gazelle_binary",
)
```

Extending gazelle - instructions

Nix

Difficulties

- **Nix language**
- Lazy and dynamic
- It is hard to predict what files are going to be needed
- Static analysis would not be feasible solution to write
- How to list abstract nix derivations file dependencies?

Nix

Attempt #1 - standard nix

- ✗ Static analysis is out of the question
- ✗ Querying nix store: completeness not guaranteed
- ✗ Inspecting derivations: hard to trace back inputs to their origin
- ✓
```
nix-instantiate -vvvvv \
| grep 'copied source'
```

Nix

Attempt #2 - patched nix

- ✗ Patching nix source code is easy, but hard to maintain
- Extending evaluation with `scopedImport`: undocumented on purpose
- ✓ Stitching bits of `lorri` with the extension got us first working version
- ✗ Code was slow, unstable and bloated

Nix

Attempt #3 - traced nix

- ✓ No need to modify evaluator nor evaluation process
- ✓ Negligible performance impact
- `strace` had all information we needed
- ✗ ...and way more
- ✓ Hit the jackpot with **fptrace**

Recap

Recap

- bazel-gazelle can do a lot of heavy lifting for your Bazel generation
- As long as you know some golang, extending gazelle is trivial
- Use **tweag/nix_gazelle_extension** for generation of nix-to-Bazel glue code
  - `default.nix` signifies what should be a package
  - Use gazelle directives to apply the ”nix preamble”
  - You can mark code which should not be updated

Next steps

Next steps

- Parametrize the `default.nix` package marker
- Improve the testing suite
- Expand documentation

Questions

Thanks!

References

- **tweag/nix_gazelle_extension**
- **Nix language**
- **Illustrious rules_nixpkgs**
- **bazelbuild/bazel-gazelle**
- **fptrace**
- **Tweag website**
- **Qarik website**