When "go build" isn't enough: Introduction to Bazel

Apr 25, 2024

Eugene Khabarov
Lead Developer, Arctic Wolf
Ottawa, ON, Canada
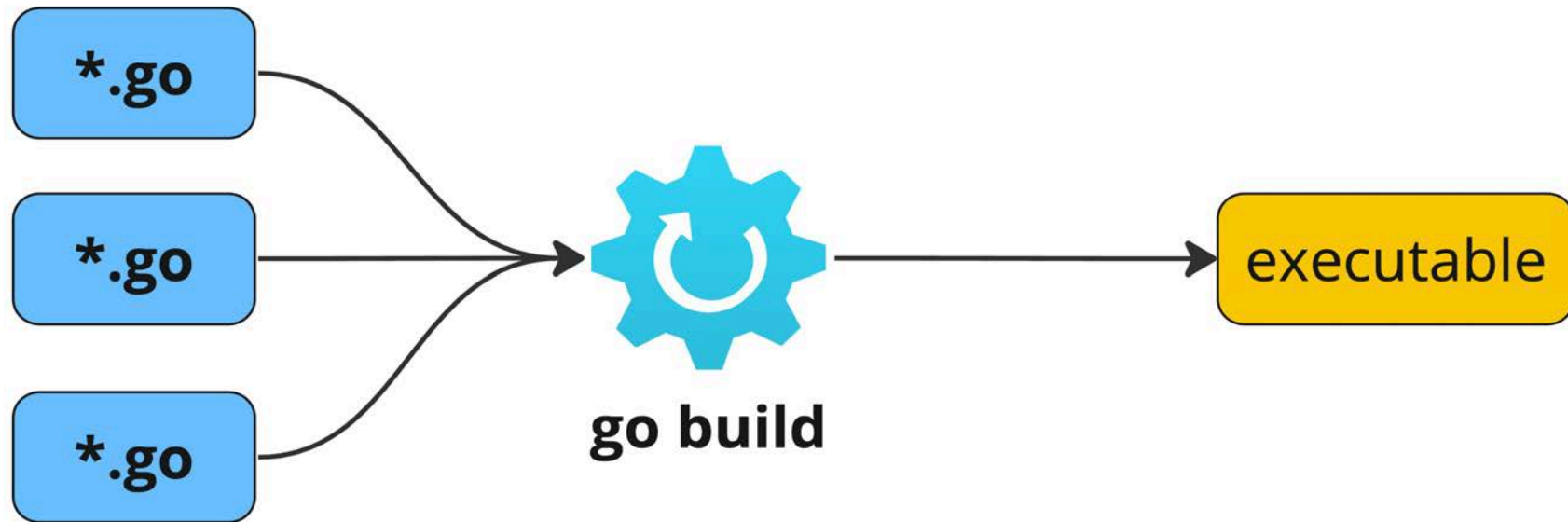
Agenda

- Build process

- Dependencies

- Bazel

2

Build and dependencies

3

What is "go build"?

- A command which compiles the packages named by the import paths, along with their *dependencies*.

- **go build** takes a list of ***.go** files as an argument and produces a executable binary.

Compile packages and dependencies (https://pkg.go.dev/cmd/go#hdr-Compile_packages_and_dependencies)   4

Do we have all of the *.go files from the beginning?

- *// go:generate <some arbitrary binary or script here>*

- make generate

- generate.sh

- etc.

5
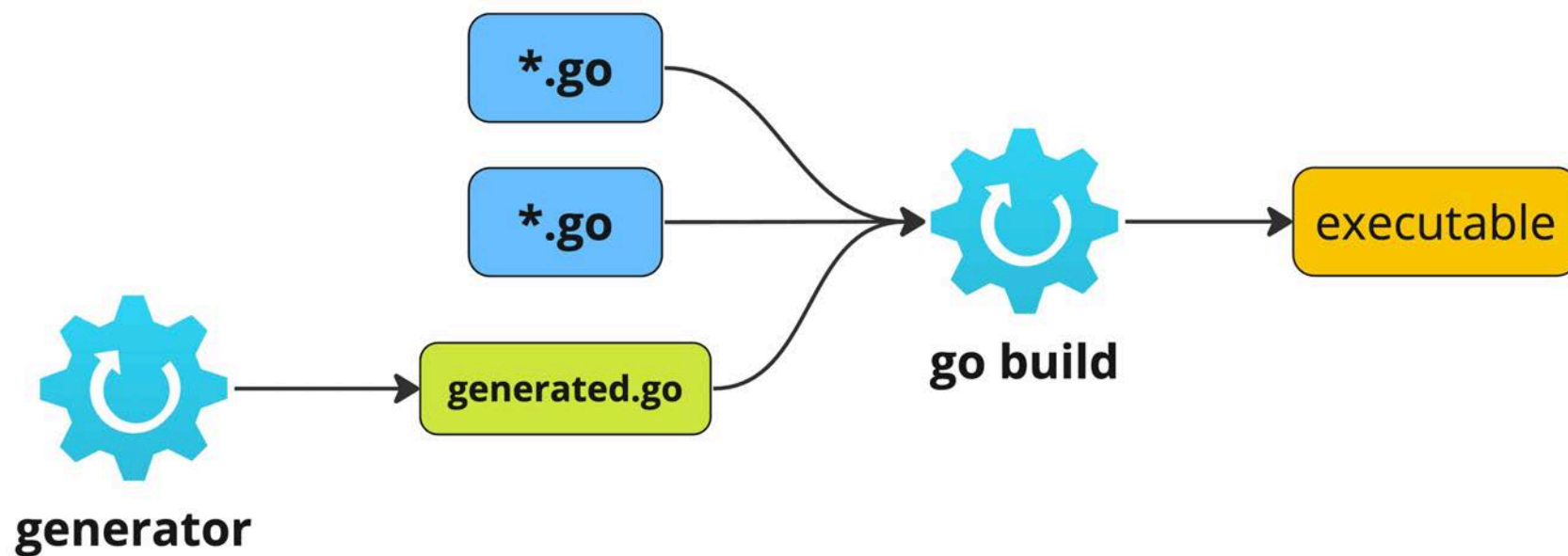
Dependencies: part I

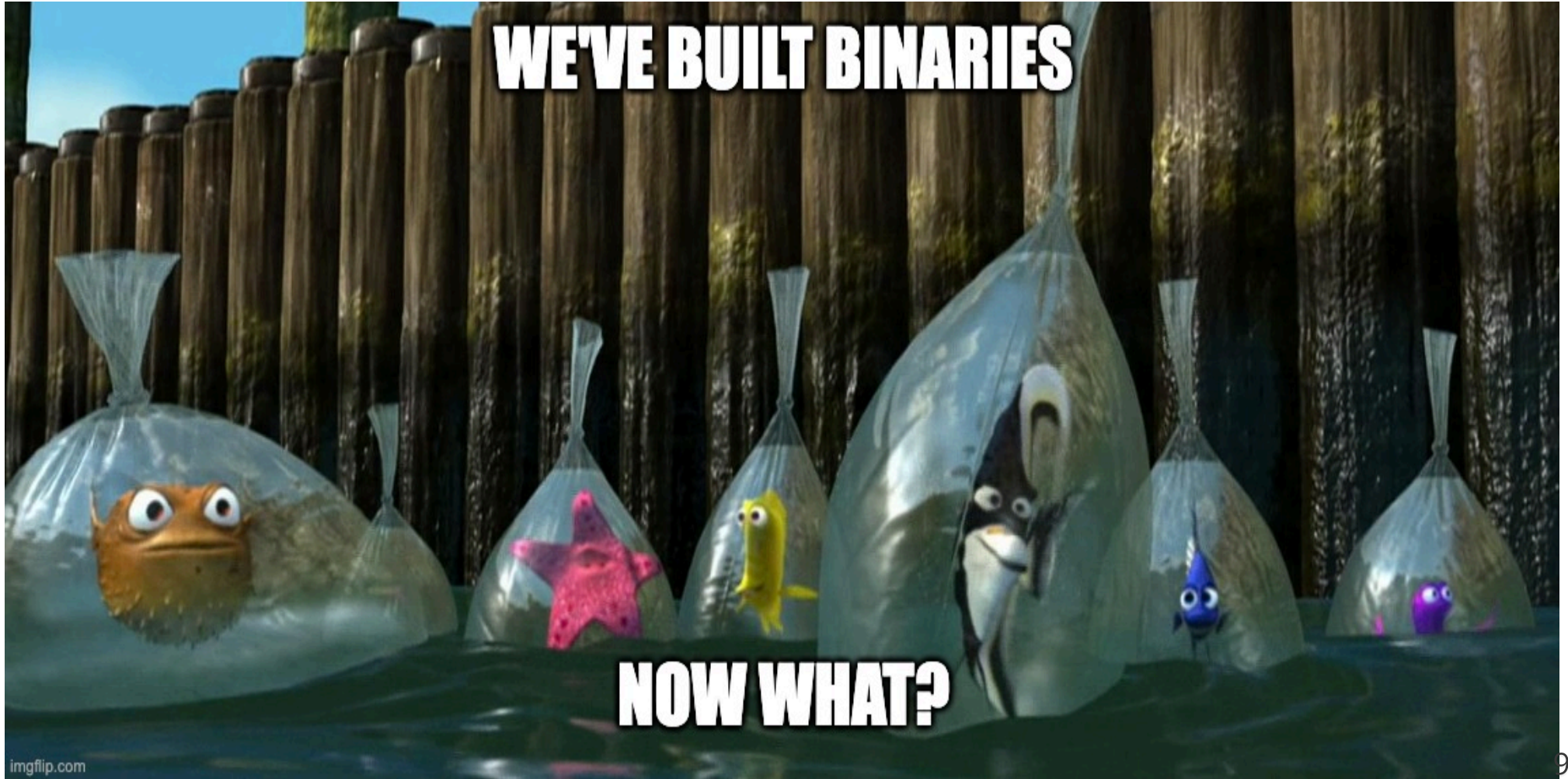- Go packages

- Go compiler

- Generators

6

Where do we run our build?

- MacOS / Linux / etc.

- amd64 / arm64 / etc.

- Local machine / CI runner / etc.

7

Dependencies: part II

- **Host machine** is a machine where we run a build

- **Target machine** is a machine we build software for
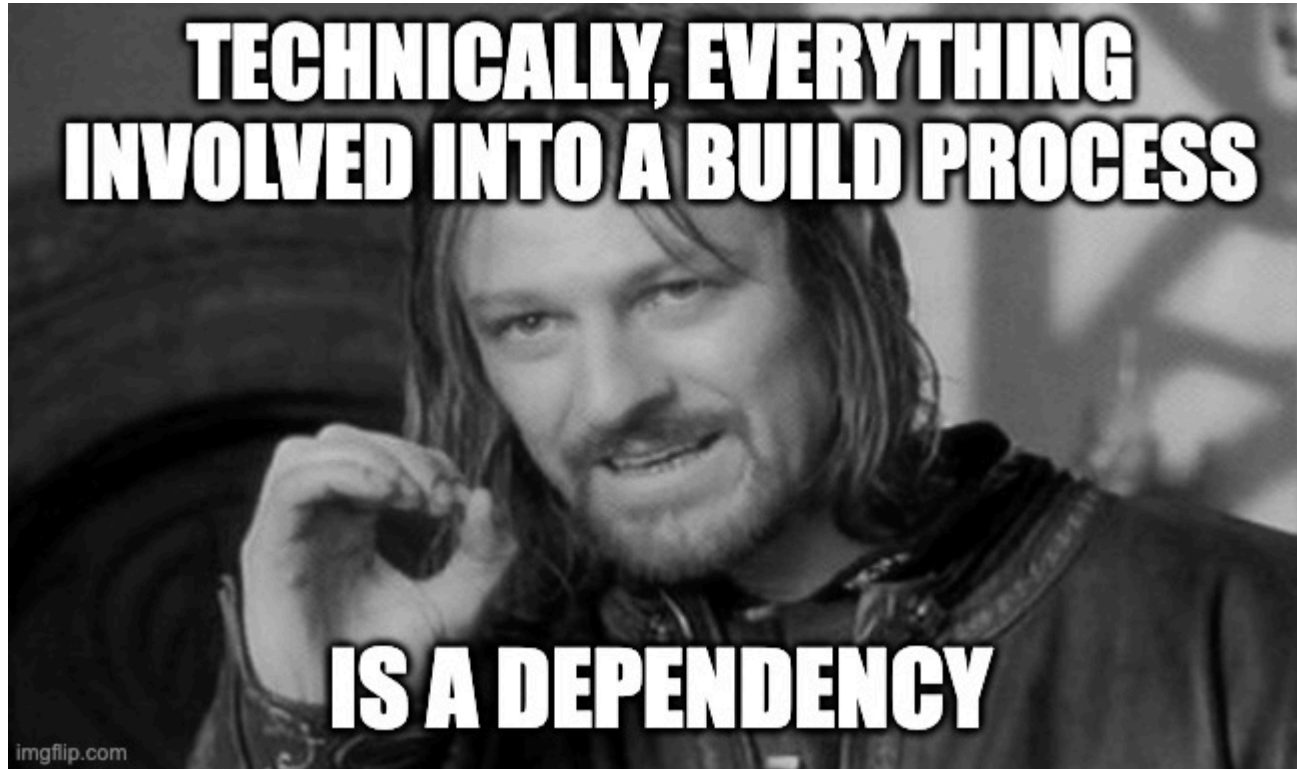
- Environment variables

8

Publishing

- Container images

- tar/zip archives: AWS lambdas

- Kubernetes manifests / Helm charts / CloudFormation templates / etc.

- etc.

10

Dependencies: part III

- Docker

- Kustomize / Ytt / Helm / any other templating tool

- AWS cli

- Make (for some automation of all above)

- etc.

11

What is the dependency, btw?

12

How do we control our dependencies?

- **Go packages**: go.mod/go.sum.

- **Go compiler**: specific version pre-installed into container image or random Go version on host machine.

- **Generators**: depends on generator.

- **Platforms**: build flags or running build on a specific platform.

- **Environment variables**: explicitly set during the build.

- **Docker / Kustomize / Ytt / Helm / AWS cli**: do we?

13

# What else should we consider during the build process?

- Is our build reproducible?

- Is our build well isolated/hermetic*? (Hello Docker!).

- When build fails, can we restart from the failure point but not from the beginning?

```
* more on this later
```

14

The output of the build process aka artifacts

- Go binaries

- Container images and/or tar/zip archives

- YAML manifests

We're going to build and publish **ready-to-deploy** artifacts, but not to deploy them.

15
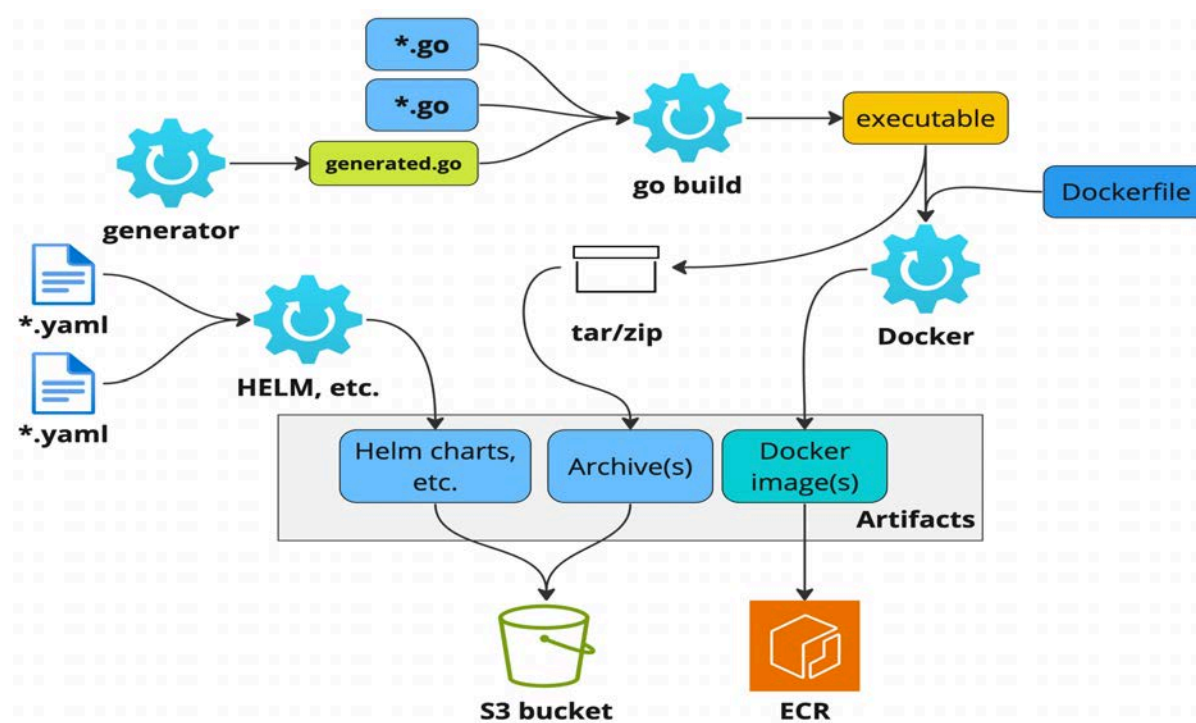
Why "go build" isn't enough?

- **go build** is just an one step of the build process.

- While what we actually need is a Build orchestration.

16

Problem scope

Automate the following process:

- Download and install **all** necessary dependencies (generators, compilers, tools, etc.)

- Build artifacts.

- Publish artifacts to ECR / S3 / etc.

Other requirements:

- Make a build as isolated as possible.

- Make a build as reproducible as possible, i.e. pin versions of **all** dependencies.

17

Bazel

18

Hello Bazel!

- An open-source build and tests tool that uses human-readable, high-level build language to define build in a declarative way.

- Aimed to build large codebases.

- Supports multi-language and multi-platforms builds

- It unifies build approaches across multiple languages and multiples toolchains.

19

Build & cache

- Parallel build: Bazel uses as many cores as it found.

- Build can be run on a local or remote machine.

- Can build everything from sources including dependencies.

- Bazel caches all downloaded dependencies and intermediate build results.

- Tracks changes in sources and rebuilds changed parts only.

20

Hermeticity & Sandboxing

Hermiticity:

When given the same input **source code** and **product configuration**, a hermetic build system always returns the same output, i.e. hermetic builds are **insensitive** to libraries and other software installed on the host machine.

Source identity:

Hermetic build systems try to ensure the sameness of inputs by using checksums to identify changes to the build's input.
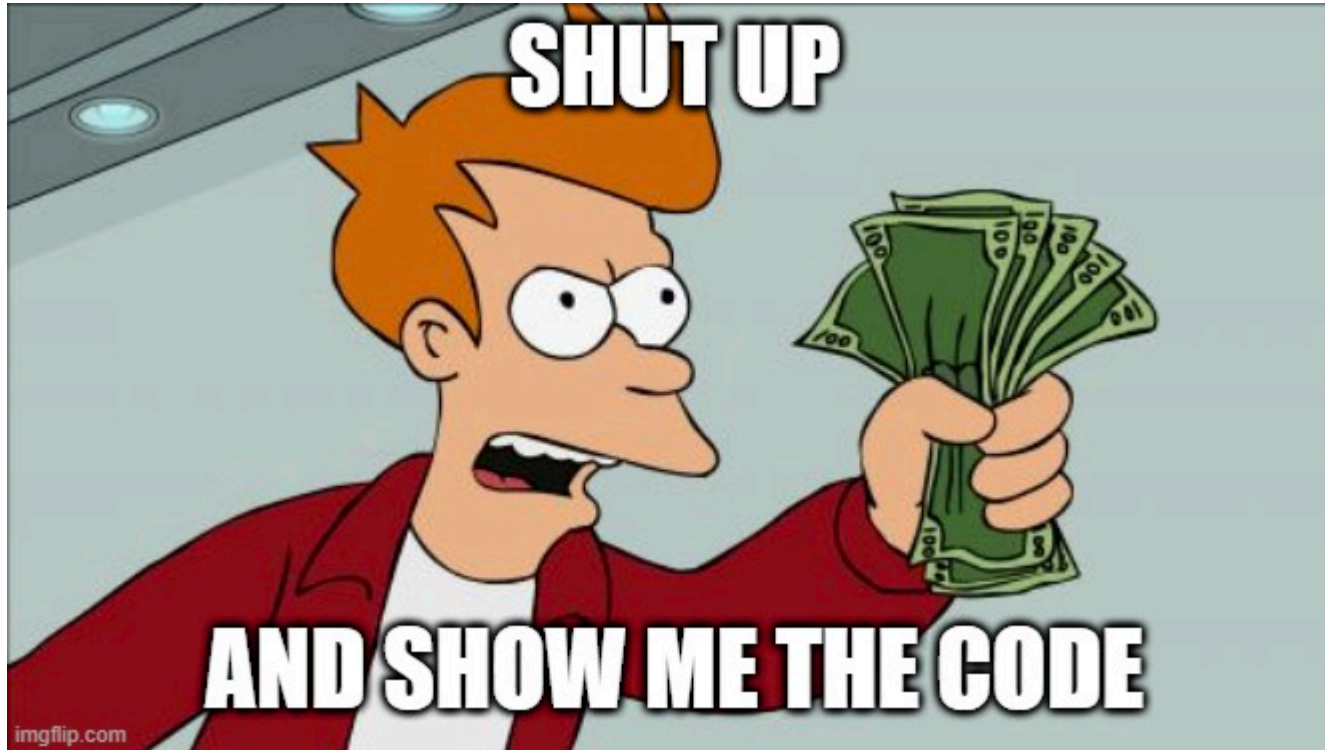
Sandboxing:

Compilers and other tools during the build have an access to explicitly defined inputs only.

bazel.build/docs/sandboxing (https://bazel.build/docs/sandboxing)

bazel.build/basics/hermeticity (https://bazel.build/basics/hermeticity)   21

It's demo time!

22

Hello world generator (main.go)

```go
package main

import "fmt"

func main() {
    fmt.Println(`
            package main

            import "fmt"

            func main() {
                fmt.Println("Hello, World!")
            }
    `)
}
```

github.com/ekhabarov/helloworld-generator (https://github.com/ekhabarov/helloworld-generator)   23

Hello world generator

```
% go build -o hw_generator main.go

   //go:generate hw_generator > hw.go

% go generate ./...

% cat hw.go
package main

import "fmt"

func main() {
    fmt.Println("Hello, World!")
}

% go build -o hello-world hw.go

% ./hello-world
Hello, World!
```

24

And with Bazel

```
% bazel run //go:hello-world
Hello, World!
```

- **run** - Bazel command.

- **//go:hello-world** - label, a unique name for a target.

- **//** - project root.

- **//go** - package.

- **:hello-world** - build target.

25

Hello world generator: Bazelified

- **WORKSPACE**: defines a project root **//** and may contain external dependencies.

- **BUILD.bazel**: defines a package like **//** or **//go**. Declares zero or more build targets for the package. In most of the cases Bazel package structure is the same as directory structure.

```
.
├── BUILD.bazel
├── WORKSPACE
└── go
    └── BUILD.bazel
```

Github Repo (https://github.com/ekhabarov/blog-code-snippets/tree/master/how-to-bazel/genereate-and-compile-go-code)

Blog post (https://ekhabarov.com/post/how-to-generate-code-with-bazel/)   26

Starlark

Starlark syntax is a strict subset of Python and Starlark semantics is almost a subset of that language. In particular, its data types and syntax for statements and expressions will be very familiar to any Python programmer. However, Starlark is intended not for writing applications but for expressing configuration.

27

WORKSPACE

- **Ruleset**: An extension for Bazel.

```
load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive")

http_archive(
    name = "io_bazel_rules_go",
    sha256 = "dd926a88a564a9246713a9c00b35315f54cbd46b31a26d5d8fb264c07045f05d",
    urls = [...],
)

load("@io_bazel_rules_go//go:deps.bzl", "go_register_toolchains", "go_rules_dependencies")

go_rules_dependencies()

go_register_toolchains(version = "1.20.3") # Go version

load("@bazel_gazelle//:deps.bzl", "go_repository")

go_repository(
    name = "hw_generator",
    importpath = "github.com/ekhabarov/helloworld-generator",
    sum = "h1:MrREQgX6I0/4cstUhbuqfALzUF3W2Nz8kVZRq6A4q+E=",
    version = "v0.0.1",
)
```

28

go/BUILD.bazel

- **Rule**: A function implementation. It takes an input and produces an output.

- **Target**: A buildable unit.

```
load("@io_bazel_rules_go//go:def.bzl", "go_binary", "go_library")

genrule(
    name = "generate_hello_go",
    outs = ["hello.go"],
    cmd = "$(execpath @hw_generator//:helloworld-generator) > $@",
    tools = ["@hw_generator//:helloworld-generator"],
)

go_library(
    name = "hello-world_lib",
    srcs = ["hello.go"],
    importpath = "github.com/ekhabarov/helloworld-generator",
)

go_binary(
    name = "hello-world",
    embed = [":hello-world_lib"],
    importpath = "github.com/ekhabarov/helloworld-generator",
)
```
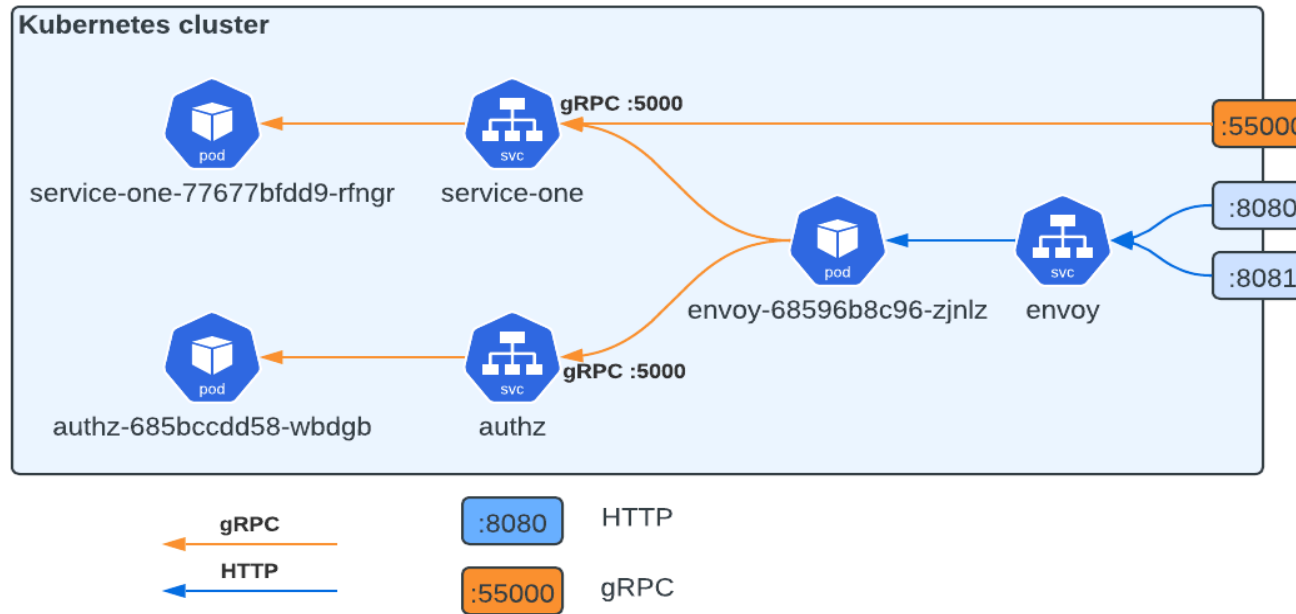
29

Build & run

```
bazel run //go:hello-world                          => runs hello-world binary
   |
   |--> bazel build //go:hello-world                => creates hello-world binary
           |
           |--> bazel build //go:hello-world_lib     => creates hello-world.a
                   |
                   |--> bazel build //go:generate_hello_go   => creates hello.go
```

30

Demo: Microservices, Kubernetes and Tilt

- build two gRPC services

- build Docker images for the services, and for Evnoy proxy

- deploy all into local k8s cluster (minikube)

ekhabarov.com/envoy (https://ekhabarov.com/envoy)   31

Labels

```
Common format
% bazel (build | test | run) //path/to/package:target

Build everything in a workspace
% bazel build //...
or
% bazel build //:all

Build everything in package "abc" recursively
% bazel build //abc/...
or
% bazel build //abc/:all

Build external dependency
% bazel build @hw_generator//...

Build one target
% bazel build //go:hello-world

Run one target
% bazel run //go:hello-world
```

32

Should we write BUILD files manually?

Partially, thanks to Gazelle, which:

- Generates BUILD files

- Keeps them up to date

- Formats BUILD files

- Manages dependencies

Manually added targets:

- Container images

- YAML manifests

- Publishing artifacts
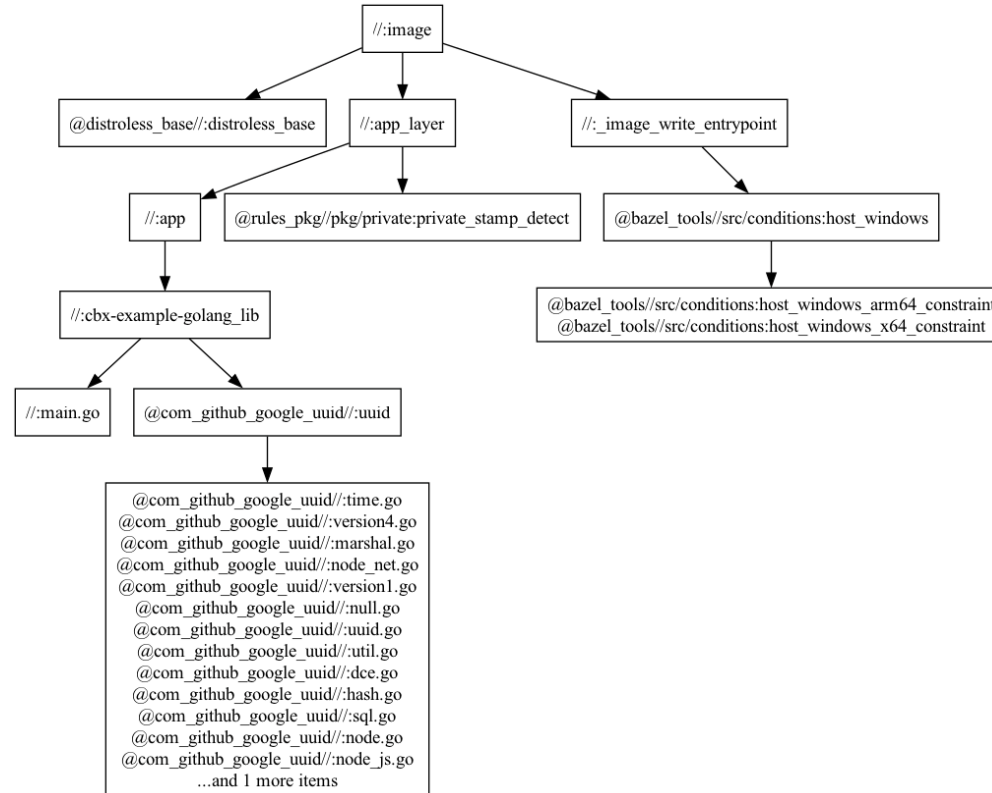
33

What else?

34

Bazel query

Some questions that query answers:

- What packages use package or tool *X*?

- Which dependencies package *X* has?

- What files are generated *foo* package?

- What rule target(s) contain file *path/to/file.go* as a source?

- Where a transitive dependency came from?

Bazel Query How-To (https://docs.bazel.build/versions/main/query-how-to.html)   35

Dependency graph


```
$ bazel query --noimplicit_deps 'deps(//:image)' --output graph | grep -v ... > graph.in
$ dot -Tpng < graph.in > graph.png
```

36

Extensibility

- **Ruleset**: An extension for Bazel.

Available rules:

- rules_go

- rules_oci

- rules_proto

- rules_ytt

- rules_erlang

- rules_haskell

- rules_kotlin

- etc.

awesomebazel.com (https://awesomebazel.com)   37

More info on Bazel

bazel.build (https://bazel.build)

awesomebazel.com (https://awesomebazel.com)

38

Thank you

Eugene Khabarov
Lead Developer, Arctic Wolf
Ottawa, ON, Canada

https://ekhabarov.com (https://ekhabarov.com)