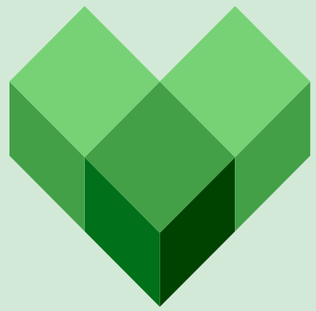
Shall we bazelize our go?



Bazel

BarnerGo 21.03.2023

Are you using Bazel already?

– no: this talk is for you

– yes: are you happy with it?

- yes: are you happy with it?
  - no: we'll peak under the hood a little
  - yes: hope you enjoyed the other talks

Scope

Why or when to use it

~~How to use it~~

About bazel...

> **{ Fast, Correct } — Choose two**
> bazel.build

Build tool/frameworks

Bazel

Make

Mage

Rake

Pants

...

Make

- `Makefile` files
- `make everything`
- Needs tools installed, host dependent

Bazel

- Workspace & `BUILD` files
- `bazel build //...`
- Fetches and sandboxes most tools, hermetic

Practical comparison 0/2

```
make --file sub/dir/Makefile test

bazel test //sub/dir:test
# or recursively everthing in sub dir:
bazel test //sub/dir/...
```

Practical comparison 1/2

```
cd sub/dir
make myapp

bazel build :myapp
# abs path also works:
bazel build //sub/dir:myapp
```

Opinionated

- `go fmt` → `buildifier`
- Build and test targets in `BUILD` files
- **All** dependencies in workspace files
  - `rules_go` for building go in bazel
  - specified go compiler version
  - any thing we would have in go.mod
- Dependencies at workspace level (go workspaces like out of the box)

Bazel and go

There are 3 *official?* toolchains:

- `go`
- `bazel`
- `blaze` (internal precursor to `bazel`)

Quick dependency example

```
# ...
go_repository(
    name = "com_github_spf13_cobra",
    importpath = "github.com/spf13/cobra",
    sum =
"h1:X+jTBEBqF0bHN+9cSMgmfuvv2VHJ9ezmFNf9Y/XstYU=",
    version = "v1.5.0",
)
```

Think of this as `go.mod` + `go.work` combined.

Quick `BUILD` example

```
go_binary(
    name = "cmd",
    srcs = glob(["*.go"]),
    deps = [
        "@com_github_spf13_cobra//:cobra",
        "//mypath/util",
    ],
)
```

Note: there's a CLI that generate this from go.mod files, let's skip the *how* for now.

Wait... what? And why?

What if...?

What if we don't need bazel?

- Small team(s)
- Few languages
- No CI/CD build time issues (<5min)
- Small code base and/or no monorepo
- No prior in house knowledge

...but what if you do?

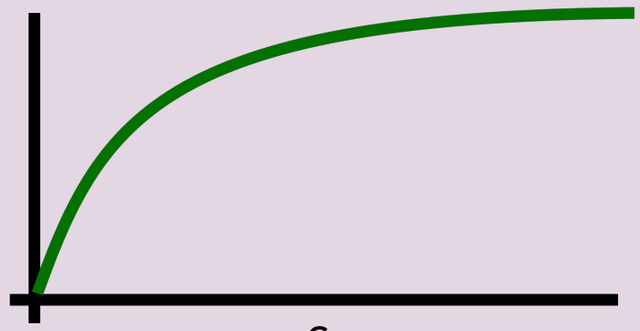
- Many teams and engineers
- Dedicated build/platform team
- Many projects/deps or/and monorepo
- Many languages (3+)
- Painfully slow CICD builds (>30min)
- Breaking library changes discovered late

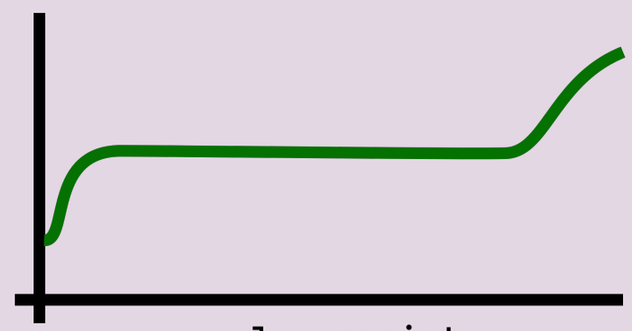
Pros and cons
Happy path...
...Honest warnings

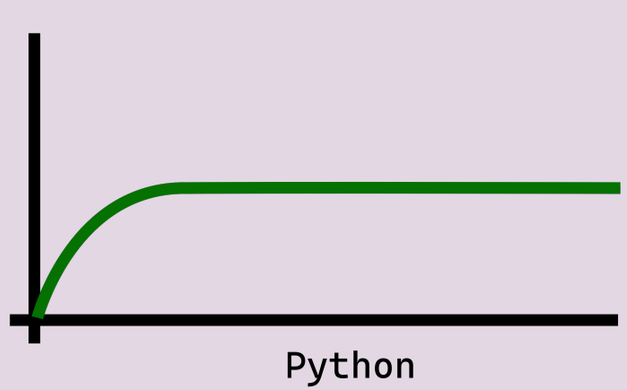
Honest warnings

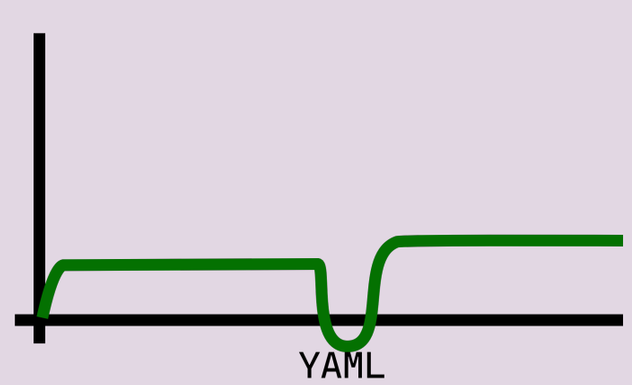
hic sunt dracones

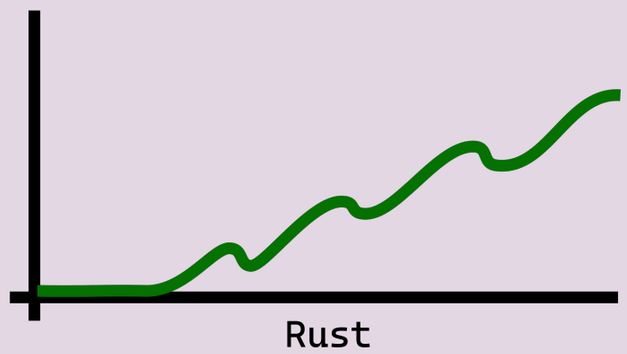
Steep learning curve

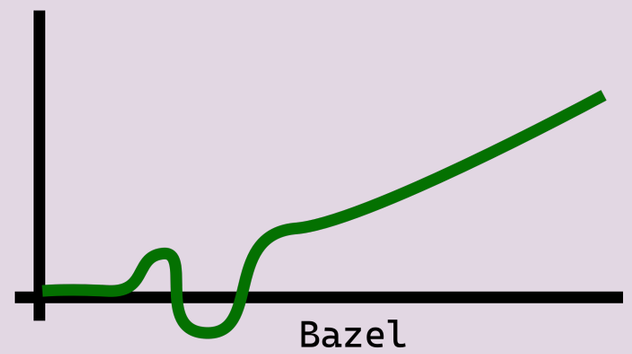












Extend with any language

- Official guides for C++, Java, iOS, Android
- Many other languages supported but examples and support may vary
- Anything else can be added

Milage may vary...

...Support can take time...

...Linting not included

Happy path

Qui audet adipiscitur

Local and cicd parity for tooling/cmds

I. Same commands all around

```
bazel build //x/y/...
bazel test //x/y/...
bazel run //x/y:z -- --some args
```

II. Sandboxing & tool fetching

Caching for large builds

Known inputs → known outputs

Building + **testing** everything:

- No cache: ~60min
- Remote cache: ~15min
- Mostly cached locally: ~5min

Dependency tree & query engine

- Fetch only what is needed for given target
- **Re-build what's affected by a change**
- Query which files uses x/net?

How we use it at Open Systems

SSE & SASE services

- Over 10K hosts worldwide
- 24/7 Network Operation Center
- Office in Hawaii
- Open positions (w/ Go)

  www.open-systems.com

- All teams on a mono repository
- Trunk based workflow
- 4 eyes workflow (code reviews)
- 5+ languages (java, go, perl, python, ...)

- 400 releasable modules, about half use bazel
- bazel for java, ts, go, python, helm, ...
- 1900 bazel `BUILD` files

- *Monopipeline* CI/CD workflow
- Currently ~2k builds/week
- Pipelines: <5minutes

2 of key benefits

Build *everything* fast with caching

Changes knowing dependents will rebuild/retest

2 of significant costs

Steep learning curve

Maintenance required to keep up to date

Bazel for go

a useful go build framework...

...not a silver bullet

References

- https://bazel.build/
- https://github.com/bazelbuild/rules_go
- https://github.com/bazelbuild/bazel-gazelle