# Introduction to Bazel

**{ Fast, Correct } — Choose two**

https://bazel.build/

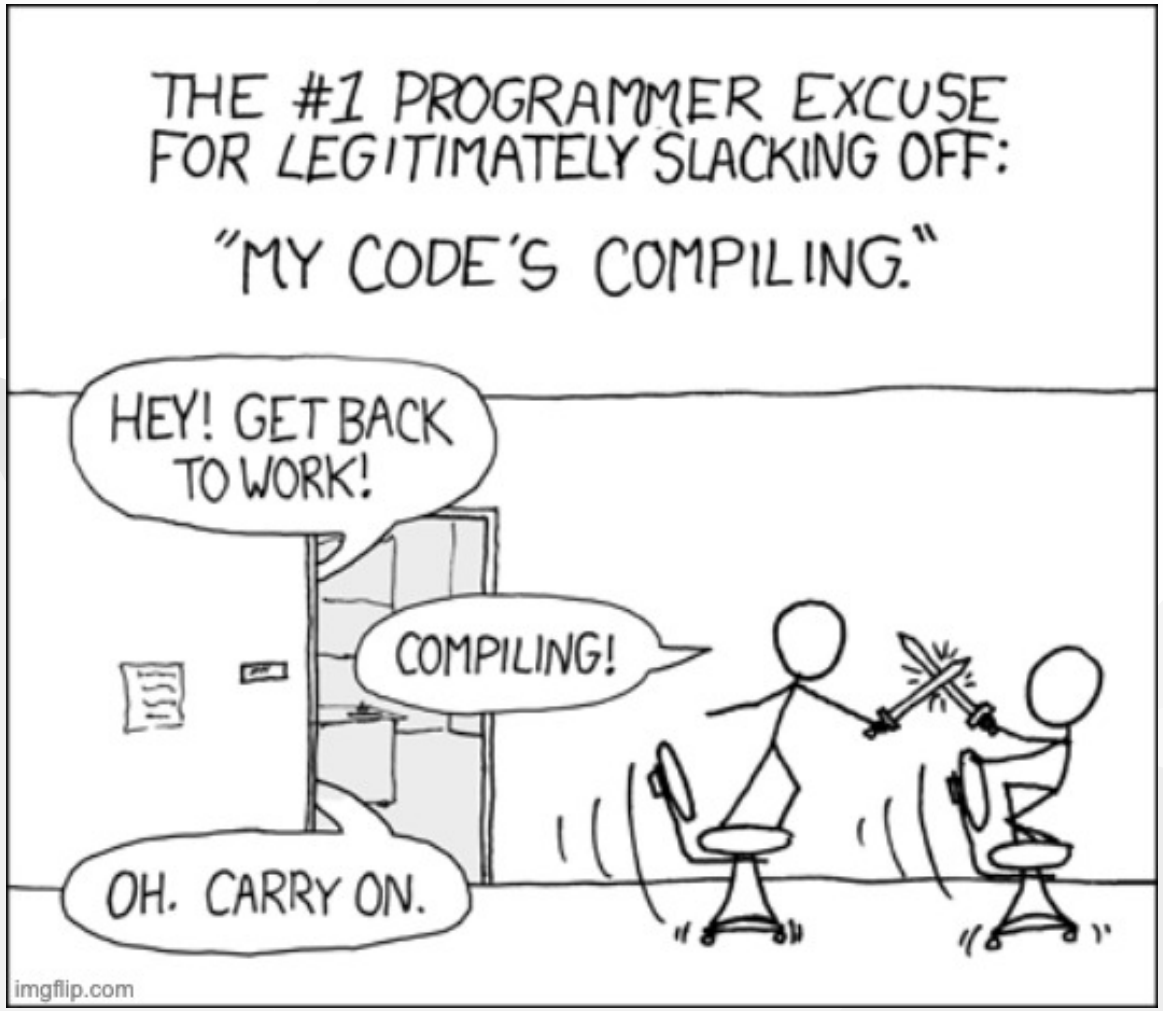
Rikito Taniguchi

# Hi! 👋

- Rikito Taniguchi (@tanishiking on github)
- Working from Japan 🗾
- Scala Space, working on Scala ecosystem
  - LSP, compiler, formatter and linter
- Bazel experience: 1.5 month
  - but I'm loving it!

# Agenda

- The "problem"
- What and Why Bazel
- Bazel tutorial along with Scala
  - How to build Scala files
  - How to use 3rd party library




# The "Problem"

- Building large applicaton is slow
- Building slowly is expensive

# Options to alleviate the problem

- Optimize Scala compilation (maybe using scalac-profiling)
- Optimize sbt build
  - -Dsbt.traces=true
  - Custom configuration
  - build cache
- Split to multi-repo
- Compile Scala Faster with Hydra - Triplequote

Still, they're not **scalable**

# Bazel for rescue!

Build system developed by Google

- **Artifact based build system**
  - ↔ Task based build system (make, ant, sbt...)
- **{Fast, Correct} Choose two**
  - ➡ Scalable even in Google scale

# {Task, Artifact}-based build system

- **Task based build system** (ant, make, maven, sbt)
  - **Imperative** set of tasks (imagine Makefile).
  - You can do pretty much anything 👍
- **Artifact based build system** (bazel, pants, buck)
  - **Declative** set of artifacts to build, deps, and limited options
  - Only build, test, and run.
  - **Your build is pure function**

[Software Engineering at Google | chapter 18]

# {Fast, Correct} choose two ✌️

How to make build a "pure function"?

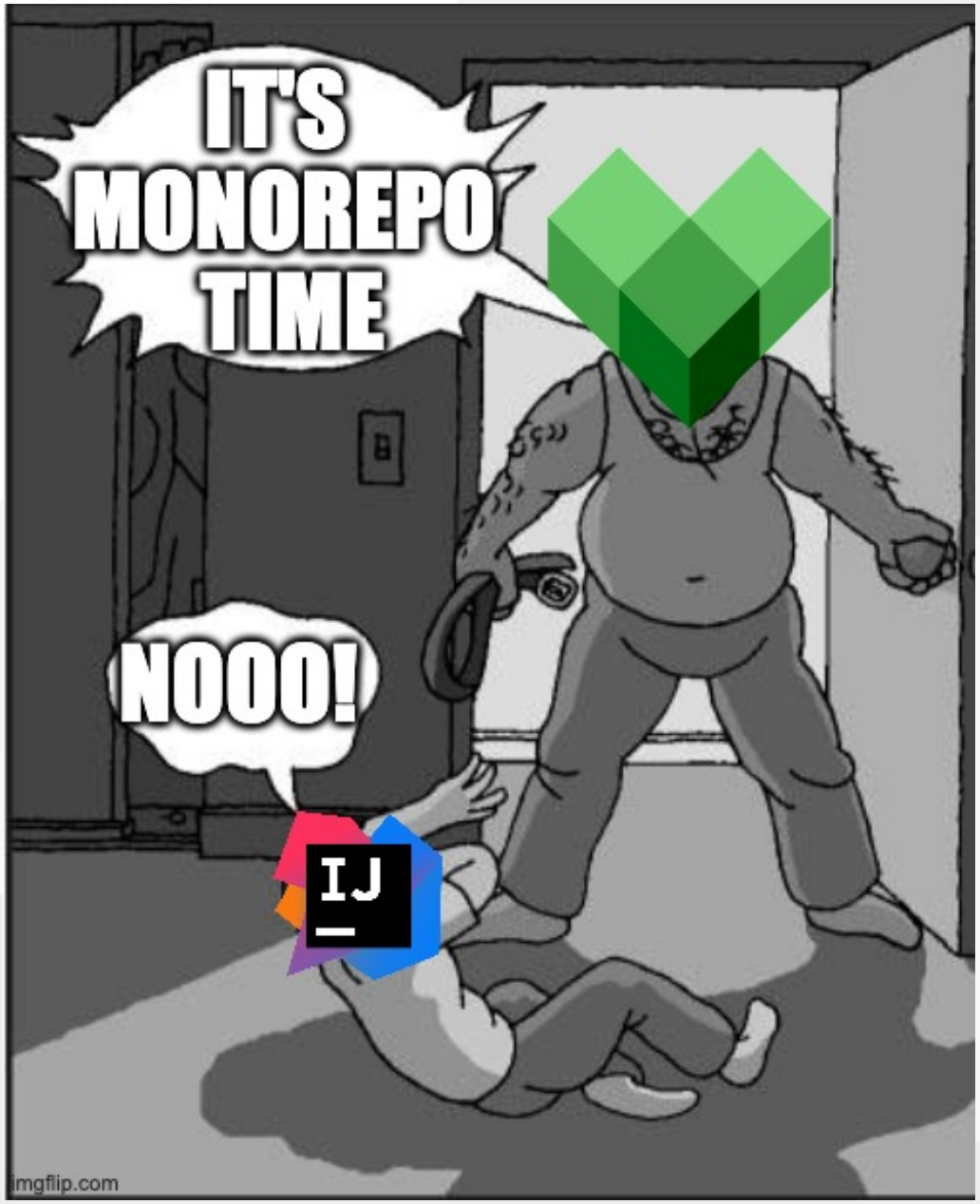
**Hermeticity**

> When given the same input source code and product configuration, a hermetic build system always returns the same output **by isolating the build from changes to the host system**

➡️ **Correct** ➡️ reliable remote build cache ➡️ **Fast**

# Dark side of Bazel

- Poor IDE support (it's getting better though...)
- [How your Monorepo breaks the IDE. And what we're doing about it. - Justin Kaeser - YouTube]
- Less flexibility
- More explicit build settings



# Is Bazel a right path?

Not sure, yet!

- We have only around 200k lines of Scala code
- but we're going mono-repo and project will grow
- Scala compile is slow for LOC...

When to use Bazel? - Earthly Blog

## Depends on how much developers willing to deal with the trade-offs

# All team members MUST learn Bazel

Otherwise...

> “ New team mebers didn't learn Bazel ... most of the members could not write Bazel-related code and they just use what there is. ”

(Japanese blog) Say goodbye to Bazel and start using make

# Questions so far?

next we're going to go through Bazel/Scala tutorial

# Bazel Tutorial for Scala

**What you'll learn**

- Bazel 101
  - What the Bazel project looks like
  - What inside `WORKSPACE` and `BUILD` files
  - What is `Label` in Bazel
- How to build jar from Scala fiels using `rules_scala`

[tanishiking/bazel-tutorial-scala]

# Install Bazel

Use Bazelisk! It reads `.bazelversion` and download Bazel executable.

[bazelbuild/bazelisk: A user-friendly launcher for Bazel.]

> Install it as the bazel binary in your PATH (e.g. copy it to /usr/local/bin/bazel). Never worry about upgrading Bazel to the latest version again

```
alias bazel="bazelisk" # I personally do
```

## bazel-tutorial-scala/01_scala_tutorial

```
|-- WORKSPACE
`-- src
    `-- main
        `-- scala
            |-- cmd
            |   |-- BUILD
            |   `-- Runner.scala
            `-- lib
                |-- BUILD
                `-- Greeting.scala
```

- `WORKSPACE` file is about getting stuff from the outside world into your Bazel project. Located at the project root.
- `BUILD` files are about what happening inside of your Bazel project

# Understand WORKSPACE

```
load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive")
# ...
http_archive(
    name = "io_bazel_rules_scala",
    sha256 = "77a3b9308a8780fff3f10cdbbe36d55164b85a48123033f5e970fdae262e8eb2",
    strip_prefix = "rules_scala-20220201",
    type = "zip",
    url = "https://github.com/bazelbuild/rules_scala/releases/download/20220201/rules_scala-20220201.zip",
)
```

https://github.com/tanishiking/bazel-tutorial-scala/blob/main/01_scala_tutorial/WORKSPACE

Basically, just copy and pasted from bazelbuild/rules_scala

# Scala files

```scala
// cat src/main/scala/lib/Greeting.scala
package lib
object Greeting { def sayHi = println("Hi!") }
```

```scala
// cat src/main/scala/cmd/Runner.scala
package cmd
import lib.Greeting
object Runner { def main(args: Array[String]) = { Greeting.sayHi } }
```

- `lib/Greeting.scala` is a library moduel that provides `lib.Greeting`.
- `cmd/Runner.scala` depends on `lib.Greeting`.

# Understand `BUILD` file for `lib`

```
# cat src/main/scala/lib/BUILD
load("@io_bazel_rules_scala//scala:scala.bzl", "scala_library")

scala_library(
    name = "greeting",
    srcs = ["Greeting.scala"],
)
```

- `scala_library` is called `rule` in Bazel that describes what to build
- An instance of `rule` is called `target`.

[document: rules_scala/scala_library.md]

# Let's build!

`bazel build <targets>`

```
❯ bazel build //src/main/scala/lib:greeting
...
Target //src/main/scala/lib:greeting up-to-date:
  bazel-bin/src/main/scala/lib/greeting.jar
```

🎉

Wait, what `//src/main/scala/lib:greeting` means!?

# Label

Label uniquely identifies a `target`. Canonical form of label looks like

**@myrepo//my/app/main:app_binary**

- **@myrepo//** - repository name to access workspace, we can omit `@myrepo` and `//` to refer same repository.
- **my/app/main** - path to the package relative to repository root.
- **:app_binary** - target name

[Labels | Bazel]

# Label

```
❯ bazel build //src/main/scala/lib:greeting
...
Target //src/main/scala/lib:greeting up-to-date:
  bazel-bin/src/main/scala/lib/greeting.jar
```

**//src/main/scala/lib:greeting**

- **//** - (abbreviated) repo name
- **src/main/scala/lib** - path to `BUILD` file (from workspace root)
- **:greeting** - target name to build

# Depends on `lib` target!

```
# cat src/main/scala/cmd/BUILD
load("@io_bazel_rules_scala//scala:scala.bzl", "scala_binary")
scala_binary(
    name = "runner", main_class = "cmd.Runner",
    srcs = ["Runner.scala"],
    deps = ["//src/main/scala/lib:greeting"],
)
```

scala_binary rule generate a jar file, and shell script to run the jar

Enumerate all dependent targets in `deps` attr

Dependencies | Bazel

# Build the binary!

Oops build failed

```
❯ bazel build //src/main/scala/cmd:runner

ERROR: .../01_scala_tutorial/src/main/scala/cmd/BUILD:3:13:
in scala_binary rule //src/main/scala/cmd:runner:
target '//src/main/scala/lib:greeting' is not visible from
target '//src/main/scala/cmd:runner'.
```

Bazel has a concept of visibility, and by default, all targets' visibility is `private`, targets in the same package can access them.

# Make `lib` visible from `cmd` package

```
 scala_library(
     name = "greeting",
     srcs = ["Greeting.scala"],
+    visibility = ["//src/main/scala/cmd:__pkg__"],
 )
```

visibility controll access grants to packages

- `//src/main/scala/cmd:__pkg__"` grants access to the package `//src/main/scala/cmd`
- `"//visibility:public"` grants access to all packages

# Build the binary! (again)

```
❯ bazel build //src/main/scala/cmd:runner
...
INFO: Found 1 target...
Target //src/main/scala/cmd:runner up-to-date:
  bazel-bin/src/main/scala/cmd/runner.jar
  bazel-bin/src/main/scala/cmd/runner
```

```
❯ ./bazel-bin/src/main/scala/cmd/runner
Hi!
```

# Tips: Wildcard

Usually build all targets by `$ bazel build //...`

- `//...` All targets in packages in the workspace.
- `//foo/...` All rule targets in all packages under foo dir

[Building multiple targets]

# Tips: bazel query

bazel query is useful to find target

```
❯ bazel query //... | grep lib
//src/main/scala/lib:greeting
```

- `bazel query //...` to list all targets in the repo
- `bazel query //... --output=location` to show the location of target definitions
- `bazel query "rdeps(//..., //src/main/scala/lib:greeting)"`
  - reverse deps of `:greeting` from `//...`

# Tutorial for `rules_jvm_external`

**What you'll learn**

- How to download external dependencies from maven repositories.
- How to depend on downloaded packages

https://github.com/tanishiking/bazel-tutorial-scala/blob/main/02_scala_maven

# rules_jvm_external

bazelbuild/rules_jvm_external is a popular ruleset to resolve and download JVM dependencies.

Download `rules_jvm_external` in `WORKSPACE` as always

```
RULES_JVM_EXTERNAL_TAG = "2.5"
RULES_JVM_EXTERNAL_SHA = "249e8129914be6d987ca57754516be35a14ea866c616041ff0cd32ea94d2f3a1"
http_archive(
    name = "rules_jvm_external",
    sha256 = RULES_JVM_EXTERNAL_SHA,
    strip_prefix = "rules_jvm_external-%s" % RULES_JVM_EXTERNAL_TAG,
    url = "https://github.com/bazelbuild/rules_jvm_external/archive/%s.zip" % RULES_JVM_EXTERNAL_TAG,
)
```

# Download JVM deps

```
# WORKSPACE
load("@rules_jvm_external//:defs.bzl", "maven_install")
maven_install(
    artifacts = [
        "org.scalameta:scalameta_2.13:4.5.13",
        "com.lihaoyi:pprint_2.13:0.7.3",
    ],
    repositories = [
        "https://repo1.maven.org/maven2",
    ],
)
```

# Use downloaded libraries

```
# cat src/main/scala/example/BUILD
scala_binary(
# ...
    deps = [
        "@maven//:com_lihaoyi_pprint_2_13",
        "@maven//:org_scalameta_scalameta_2_13",
    ],
)
```

“ The default label syntax for an artifact `foo.bar:baz-qux:1.2.3` is `@maven//:foo_bar_baz_qux`
https://github.com/bazelbuild/rules_jvm_external#usage ”

# Tips: find library's label

bazel query again!

Enumerate all targets under `@maven` repo, and grep `pprint`

```
❯ bazel query @maven//... | grep pprint
@maven//:com_lihaoyi_pprint_2_13
@maven//:com_lihaoyi_pprint_2_13_0_7_3
```

# Build it!

```
❯ bazel build //src/main/scala/example:app
Target //src/main/scala/example:app up-to-date:
  bazel-bin/src/main/scala/example/app.jar
  bazel-bin/src/main/scala/example/app

❯ bazel-bin/src/main/scala/example/app "object main { println(1) }"
Source(
  stats = List(
    Defn.Object(
      ...
```

## Now you learnt all Bazel basics 🎉

# Wanna learn more?

- Bazel getting started
  - Recommend to skim through **Java tutorial** and **Build concepts**
- bazelbuild/rules_scala
- bazelbuild/rules_jvm_external
- tanishiking/bazel-playground
  - You can find my Bazel example projects 😄
- Software Engineering at Google, chapter 18
  - To learn the philosophy of Bazel

# Topics I didn't cover

- Target granularity and trade-offs
  - read How to choose the right build unit granularity | by Natan Silnitsky | Wix Engineering | Medium
- Bazel devtools (attached some links in the following slide)
- Remote Caching
- Remote Execution

# Interesting Bazel talks and articles

- Awesome Bazel | awesome-bazel
- How to successfully migrate to Bazel from Maven or Gradle. (Natan Silnitsky, Israel - Youtube
- When to use Bazel? - Earthly Blog
- How to choose the right build unit granularity | Medium
- A Bable in Bazel Blog posts about Bazel internal

## Bazel dev tools

- IntelliJ with Bazel
  - Bazel IDE for IntelliJ, developed by Jetbrains + Bazel team
- Bazel - Visual Studio Marketplace
  - Syntax highlight + format + lint
- bazel-stack-vscode - Visual Studio Marketplace
  - Experimental IDE for VSCode
- JetBrains/bazel-bsp required for Scala IDE work with Bazel
- buildtools/buildifier Bazel formatter and linter
- Gazelle Bazel build file generator, Scala is not yet supported

# Home - BazelCon 2022 is around the corner!