Introduction to Bazel

{ Fast, Correct } — Choose two

https://bazel.build/

Rikito Taniguchi

What you will learn

- Why we're migrating to Bazel
    - Is this really worth it?
- Pros and Cons of Bazel (over sbt)
- Why you NEED TO LEARN Bazel
- Bazel basic concepts
- How to write Bazel build settings for Scala

Agenda

- Reminder of the "problem"
- Bazel Concept
- Bazel and Scala tutorial
- Know the trade-offs

Reminder of the Problem

- Building large applicaton is slow
- Building slowly is expensive

Mitigate the slow compilation by...

- Profiling and Speeding up Scala compilation
  - Speeding Up Compilation Time with scalac-profiling
- More finer-grained dependencies in build.sbt
  - Improve build speed using sbt's custom Configuration - xuwei-k's blog (Japanese blog)

```
val TestShared =
  Configuration.of("TestShared", "test-shared") extend Compile
```

Still, build time grows per LOC

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**
  - **Imperative** set of tasks (imagine Makefile).
  - You can do pretty much anything.
- **Artifact based build system**
  - **Declative** set of artifacts to build, deps, and limited options
  - Only build, test, and run.

Software Engineering at Google | chapter 18

{Fast, Correct} choose two ✌️

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...)
- More build settings
- Explicit dependency management (toilsome)

Is Bazel a right path?

Not sure, yet!

Bazel is not the only option

- Split to multi-repo
- Stick with sbt

When to use Bazel? - Earthly Blog

All team members MUST learn Bazel

More build configurations (than sbt)
➡️ everyone have more opportunity to write build settings
➡️ All developers 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?

Bazel Tutorial for Scala

What you'll learn

- The essential building blocks of Bazel
    - 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 checks `.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

Terminology

```
|-- WORKSPACE                                    \
`-- src                                          |
    `-- main                                     |
        `-- scala                                |> workspace
            |-- cmd                 \            |    a.k.a.
            |   |-- BUILD           |> package   |  repository
            |   `-- Runner.scala    /            |
            `-- lib                 \            |
                |-- BUILD           |> package   |
                `-- Greeting.scala /             /
```

- The whole directory to build with Bazel is called `workspace`
- A `package` is a collection of related files and a `BUILD` file

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

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

```
// 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 defined in `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"],
)
```

Tutorial for `rules_jvm_external`

What you'll learn

- How to download external dependencies from maven repositories.
- How to use it from packages.

rules_jvm_external

bazelbuild/rules_jvm_external is a

Target granularity

Tips and Tricks

External Resources

- basics
  - Bazel getting started
  - bazelbuild/rules_scala
  - bazelbuild/rules_jvm_external
- For more information
  - Software Engineering at Google, chapter 18
  - How to successfully migrate to Bazel from Maven or Gradle. (Natan Silnitsky, Israel - Youtube

Bazel related tools

- IntelliJ with Bazel
- JetBrains/bazel-bsp
- Gazelle
- Buildifier