CHAPTER 1

Introduction

Welcome to the world of Bazel!

In case you haven't heard about it, Bazel is the open source version of the build system used at Google (Alphabet). To give a sense of scale, Bazel was designed to solve some of the core problems of building at Google, namely, having to build literally *millions* of lines of code across a multitude of languages, efficiently and correctly, for multiple platforms (e.g., server, mobile, desktop) and different hardware architectures.

While the build system was initially internal to Google, it was released to open source a few years ago. Since that time, it has continued to evolve into a high-performance, powerful, yet simple build system for production-level needs.

What This Book Is

Beginning Bazel is meant as a gentle and practical introduction into using the Bazel build system. As you progress through the book, you will learn the basics of Bazel through a series of targeted examples.

These examples are aimed at teaching the core concepts and constructs of Bazel, including how to set up some basic build targets, construct and cultivate your workspace to pull in new language rules, and easily build both command line and mobile applications within the same project across multiple languages.

Through the course of this book, you will build examples in various languages, tying them together in a cohesive fashion and generating working binaries that could run on a server and on mobile (with examples covering both Android and iOS).

What This Book Is Not

Beginning Bazel is not a comprehensive reference manual. While you will learn some of the core commands for Bazel, there are *many* options and avenues that are not covered

© P.J. McNerney 2020
P.J. McNerney, *Beginning Bazel*, https://doi.org/10.1007/978-1-4842-5194-2_1

in this book (perhaps a future sequel will explore some of the areas). Fortunately, Bazel's documentation is excellent: `https://docs.bazel.build`. This has the information on latest and greatest advancements happening for Bazel.

Also, while Bazel is able to build most languages, this text only covers a very small fraction of them. Fortunately, the patterns that you will learn in this book are applicable across most of the languages that you are likely to encounter and use with Bazel.

New language rules are popping up all the time, so it is worthwhile to check out the main GitHub organization at `https://github.com/bazelbuild`. Additionally, Awesome Bazel (`www.awesomebazel.com`) is a great site for a curated set of Bazel rules and very worthwhile to check out to see some fun new possibilities for the language.

Features of Bazel

One of the chief goals of Bazel is to make sure that your builds are hermetic, that is, that the build dependencies (including both dependent libraries and build tools) are well known and independent of anything that may or may not be installed on any given machine. Ideally, any build can be reproduced using only the tools within the given project's workspace.

To this end, Bazel takes special care to ensure that you are explicitly specifying all of your dependencies and eschewing any "magic" in creating your build. Some might object that this is removing a degree of convenience. However, in reality this explicit specification allows both Bazel and the user to reason intelligently about the builds and provide tools to help diagnose and fix issues as they occur.

Bazel has many features that make it attractive as a build system:

- High-level, extensible build language
- Explicit dependency management
- Advanced visibility rules
- Explicit workspace management
- Remote build execution and caching
- Build dependency analysis
- Fast, correct builds

High-Level Build Language

Bazel provides a very simple yet powerful set of constructs. These include (but are not limited to)

- Commands (such as *build* and *test*)
- Rules (e.g., for handling different languages)
- Packages (to collate a set of rules and dependencies together)
- Workspaces (to define the working files, outputs, and dependencies of your project)

Additionally, Starlark (formerly Skylark) is Bazel's build language (inspired by Python). Starlark can further extend Bazel to create new language rules, macros to assist in development, and so on.

Explicit Dependency Management

As previously mentioned, Bazel requires explicit dependency declaration. There is no proverbial "free lunch" with Bazel as it favors being explicit over any kind of implicit "magic" (e.g., the location of the header files in C++). When creating a build target (e.g., a library), you are required to specify each of the files (or some directive collating all of the files); if you don't specify it, Bazel will not see it.

Additionally, as you depend upon other targets (e.g., another library), this dependency must be defined explicitly; otherwise, your build will likely fail. In the limit, this explicit declaration of dependencies from one target to another forms a directed dependency graph.

Finally, the directed dependency graph in Bazel is and always must be a *directed acyclic graph.* That is, there are no cycles allowed within a Bazel dependency graph. This is important when attempting to create coherent builds, since cycles in the build tree imply the need for some kind of heuristic to break the cycle (or let the cycle break the build). Attempting to create a cycle within a dependency graph and then build against it will immediately cause Bazel to break the build and warn you of the error.

Advanced Visibility Features

One of the best features of Bazel is the ability to limit the visibility of your packages and targets. That is, you can effectively reduce the scope of what packages can actually depend upon your build targets. While similar features exist in languages like Java (i.e., package visibility) and C++ (i.e., namespaces), Bazel creates the ability to constrain visibility of dependencies to any language.

Explicit Workspace Management

Similar to dependency graph management, Bazel also gives you the ability to fully specify the dependencies of your workspace on any other dependencies, including external repos. This gives you the ability to pull in code, files, and so on from other sources, often through specific versions of the external dependencies. This helps provide guarantees of correctness while still giving you flexibility.

Remote Build Execution and Caching

Although Bazel executes locally by default, Bazel also allows you to set up a distributed build system with intelligent caching. This capability is incredibly useful for speeding up individual builds as well as helping to accelerate development across an entire team.

Build Dependency Analysis

Another powerful feature is the ability to analyze a build target’s dependencies. For anyone who has ever tried to introspect into a build product and asked, “How did *that* get in there?” Bazel’s ability to understand a build target’s dependency graph will come as a welcome tool. This is incredibly useful for simplifying dependencies, optimization, and so on.

Fast, Correct Builds (and Tests)

While all the other features are grand in their own right, together they help to provide the most important feature of all for Bazel: efficient and reliable builds (and, consequently, tests). At the end of the day, the purpose of a build system is to transform code and data into working applications in a speedy and correct fashion.

Bazel utilizes its many features to create a coherent and optimized method of building products. In addition, it has an intelligent caching system to ensure that rebuilding (since development is mostly all about rebuilding) is quick and correct, with little need for cleaning.

When all is said and done, the best feature of Bazel is that it works quickly, simply, and correctly. You can put together a simple Bazel project, execute it, and then easily extend it over time.

Who This Book Is For (and Possibly Not For)

Beginning Bazel is aimed at introducing Bazel to everyone. The degree of utility you get out of Bazel, however, will largely be determined by what kinds of problems you are trying to solve.

As indicated at the beginning of this chapter, Bazel was originally designed to solve the problems around efficiently and correctly building a massive code base for multiple languages, platforms, and architectures. However, Bazel also scales really nicely, from the simplest application to a full-stack set of microservices and mobile applications.

Indeed, Bazel may be most useful if you…

- …are starting from scratch and want a build system that is going to scale with your needs
- …want to coherently build and depend upon multiple languages
- …want out-of-the-box support for defining and running tests
- …want to easily build against multiple architectures
- …want intelligent caching of build products
- …want deterministic outputs every time you build
- …want a production-level build system
- …are willing to operate within the boundaries of Bazel

This last point may seem a bit strange; however, Bazel is an opinionated build system. In order to ensure the guarantees of speed and correctness, it will actively prevent you from doing counterproductive things (e.g., circular build dependencies). Additionally, Bazel operates best when it is the primary build system. While it *can* work with other build systems (although this is outside the scope of this book), you are going to maximize the power and utility of Bazel by using it everywhere in your project.

To that end, it is worthwhile to point out that Bazel might not be for everyone. In particular, you *might* not find that much utility in Bazel under the following situations:

- You are only dealing with a single language (e.g., Java, Kotlin) for a specific purpose (e.g., server-side programming, Android). In this case, you might find existing tools (e.g., Gradle) may be just fine for your needs.
- You have a single, small project that is focused only on a single architecture with limited requirements on additional libraries (e.g., programming for iOS). Again, you might find that existing tools (e.g., Xcode project) are fine.
- You are already happy with your existing build system.

On this last point, you may already have a perfectly good build system, in which case, Bazel may just be a curiosity.

Additionally, you may already be an expert in Bazel; in this case, this book may not provide that much utility for you (in which case, you might want to give it to a friend to share your love of Bazel).

However, for everyone else, you might have just found the best build system for your needs.