WIKIPEDIA

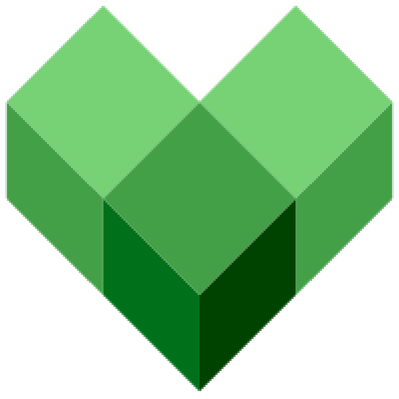
Bazel (software)

Bazel is a free software tool that allows for the automation of building and testing of software.[2] The company Google uses the build tool *Blaze* internally[3] and released an open-sourced part of the Blaze tool as Bazel, named as an anagram of Blaze.[4] Bazel was first released in March 2015 and achieved beta status by September 2015.[5]

Similar to build tools like Make, Apache Ant, or Apache Maven,[2][4] Bazel builds software applications from source code using a set of rules. Rules and macros are created in the Starlark (https://docs.bazel.build/versions/master/skylark/language.html) language (previously called Skylark[6]), a dialect of Python.[4] There are built-in rules for building software written in the programming languages of Java, C, C++, Go, Python, Objective-C and Bourne shell scripts.[4][5] Bazel can produce software application packages suitable for deployment for the Android and iOS operating systems.[7]

In designing Bazel, emphasis has been placed on build speed, correctness, and reproducibility.[2][4] The tool uses parallelization to speed up parts of the build process.[4] It includes a *Bazel Query* language that can be used to analyze build dependencies in complex build graphs.[4]

Bazel



Developer(s)	Google
Initial release	March 2015
Stable release	4.0.0 / 21 January 2021[1]
Repository	github.com/bazelbuild/bazel (https://github.com/bazelbuild/bazel)
Written in	Java[2]
Operating system	Cross-platform
License	Apache License 2.0
Website	bazel.build (https://bazel.build)

Contents

Rationale

One of the goals of Bazel is to create a build system where build target inputs and outputs are fully specified and therefore precisely known to the build system.[7] This allows a more accurate analysis and determination of out-of-date build artifacts in the build system's dependency graph. Making the dependency graph analysis more deterministic leads to potential improvements in build times by avoiding re-executing unnecessary build targets. Build reliability is improved by avoiding errors where build targets might depend on out-of-date input artifacts.

To achieve more accurate dependency graph analysis, Bazel uses content digests rather than file-based timestamps. File timestamps are commonly used to detect changes in tools like Make or Apache Ant. Timestamps can be problematic when builds are distributed across multiple hosts due to issues with clock synchronization.[8] One of Bazel's goals is to enable distributed and parallel builds on a remote cloud infrastructure. Bazel is also designed to scale up to very large build repositories which may not be practical to download to an individual developer's work machine.[9]

Bazel provides tooling which helps developers to create bit-identical reproducible build outputs. Bazel's implemented rules avoid typical pitfalls such as embedding timestamps in generated outputs to ensure content digest matches. This in turn allows the build system to reliably cache (memoize) the outputs of intermediate build steps. Furthermore, reproducible build makes it possible to share intermediate build results between teams or departments in an organization, using dedicated build servers or distributed caches. Bazel therefore is particularly well-suited for larger organizations and software projects that have significant number of build dependencies. A deterministic build and an ability to precisely analyze build input and output artifacts across the dependency graph lends itself to parallel execution of build steps.

Starlark language

Bazel is extensible with its custom Starlark programming language. Starlark uses a syntax which is a subset of the syntax of the Python programming language. Starlark however doesn't implement many of Python's language features, such as ability to mutate collections or access the file I/O, in order to avoid extensions that could create side-effects or create build outputs not known to the build system itself. Such side-effects could potentially lead to incorrect analysis of the build dependency graph.

Bazel was designed as a multi-language build system. Many commonly used build system are designed with a preference towards a specific programming language. Examples of such systems include Ant and Maven for Java, Leiningen for Clojure, sbt for Scala, etc. In a multi-language project, combining separate build systems and achieving the build speed and correctness benefits described above can be difficult and problematic.

Bazel also provides sand-boxed build execution. This can be used to ensure all build dependencies have been properly specified and the build does not depend, for example, on libraries installed only locally on a developer's work computer. This helps to ensure that builds remain portable and can be executed in other (remote) environments.

Build systems most similar to Bazel are Pants,[10] Buck, and Please.[11][12] Pants and Buck both aim for similar technical design goals as Bazel and were inspired by the Blaze build system used internally at Google. Blaze is also the predecessor to Bazel. Bazel, Pants, Buck, and Please adopted Starlark as BUILD file parser, respective its BUILD file syntax. Independently developed build systems with similar goals of efficient dependency graph analysis and automated build artifact tracking have been implemented in build systems such as tup.[13]

Sandbox

One of the key features that differentiate Bazel from other build systems is the use of a *sandbox* for compilation steps. When Bazel performs separate compilation, it creates a new directory and fills it with symlinks to the explicit input dependencies for the rule. For languages like C/C++, this provides a significant safety net for the inclusion of header files: it ensures that the developer is aware of the files that are used in compilation, and it prevents the unexpected inclusion of a similarly-named header file from another include directory.

This sandbox approach leads to issues with common build tools, resulting in a number of workarounds required to correctly compile code under different architectures. For example, when performing separate compilation for Mac/Darwin architectures, the compiler writes the input paths into SO and OSO symbols in the Mach-O binary, which can be seen with a command like `nm -a mybinary | grep SO`. These paths are needed for finding symbols during debugging. As a result, builds in Bazel must correct the compiled objects after the fact, trying to correct path-related issues that arose from the sandbox construction using flags like `-fdebug-prefix-map` and `-oso_prefix`, the latter having become available in XCode 11.0. Similar handling needs to take place in linking phases, rewriting the rpath values in shared object libraries with a command like `install_name_tool`.[14]

Logo

Since Bazel's initial release the logo was a green letter "b" stylized into a stem of a basil plant with two leaves.

On July 5, 2017, the Bazel Blog announced a new logo,[15] consisting of three green building blocks arranged to shape a heart.

Old Bazel logo

See also

- List of build automation software
- Monorepo

References

1. "Releases · bazelbuild/bazel" (https://github.com/bazelbuild/bazel/releases). *GitHub*.
2. Yegulalp, Serdar (Sep 11, 2015). "Google open-sources language-agnostic, scalable software tool" (https://www.infoworld.com/article/2983495/development-tools/google-open-sources-language-agnostic-scalable-software-tool.html). *InfoWorld*. Archived (https://web.archive.org/web/20171025022213/https://www.infoworld.com/article/2983495/development-tools/google-open-sources-language-agnostic-scalable-software-tool.html) from the original on 25 October 2017. Retrieved 25 June 2016.
3. Beyer, Betsy; Jones, Chris; Petoff, Jennifer; Murphy, Niall Richard (23 March 2016). *Site Reliability Engineering: How Google Runs Production Systems* (https://books.google.com/books?id=_4rPCwAAQBAJ&q=Bazel+build&pg=PA90). "O'Reilly Media, Inc.". p. 90. ISBN 9781491951187. Retrieved 25 June 2016.
4. Bolton, David (27 April 2015). "Bazel, Google's Open Source Build System - The New Stack" (https://thenewstack.io/bazel-googles-open-source-build-system/). *thenewstack.io*. The New Stack. Archived (https://web.archive.org/web/20171024153418/https://thenewstack.io/bazel-googles-open-source-build-system/) from the original on 24 October 2017. Retrieved 25 June 2016.
5. Daws, Ryan. "Google's software build tool Bazel heads into beta" (https://www.developer-tech.com/news/2015/sep/10/googles-software-build-tool-bazel-heads-beta/). *www.developer-tech.com*. Developer Tech. Archived (https://web.archive.org/web/20171023140229/https://www.developer-tech.com/news/2015/sep/10/googles-software-build-tool-bazel-heads-beta/) from the original on 23 October 2017. Retrieved 25 June 2016.
6. "Starlark - Bazel" (https://blog.bazel.build/2018/08/17/starlark.html). *blog.bazel.build*. Retrieved 2018-10-18.
7. "FAQ - Bazel" (https://bazel.build/faq.html). *bazel.build*. Retrieved 25 June 2016.
8. "What's Wrong With GNU make?" (http://www.conifersystems.com/whitepapers/gnu-make/).

9. Nathan York (23 September 2011). "Build in the Cloud: Distributing Build Steps" (https://google-engtools.blogspot.com/2011/09/build-in-cloud-distributing-build-steps.html). *google-engtools.blogspot.com*.
10. "Pants: A fast, scalable build system" (http://www.pantsbuild.org/index.html).
11. "Buck: A high-performance build tool" (https://buckbuild.com/).
12. Please FAQ (https://please.build/faq.html)
13. Mike Shal (2009). "Build System Rules and Algorithms" (http://gittup.org/tup/build_system_rules_and_algorithms.pdf) (PDF).
14. "tools/cpp/osx_cc_wrapper.sh" (https://github.com/bazelbuild/bazel/blob/4dfc83d5f11e9190e9e25dee4c7dc2a71cd7b8fd/tools/cpp/osx_cc_wrapper.sh#L84-L102). Github.
15. Steren Giannini (5 July 2017). "A new logo and homepage for Bazel" (https://blog.bazel.build/2017/07/05/new-logo-and-homepage.html).

External links

- Official website (https://bazel.build)