License compliance checking with Bazel - PRD

Status: Draft
Last updated: 22 Feb.. 2019
Authors: aiuto@google.com

NOTE: This is a public copy of a Google internal document gathering the requirements for a flexible license compliance checking system. We have been designing and implementing from these requirements and have reached a point where our legacy licensing checks can soon be turned down.

Also note that any implementation is intended to be in a distinct repo, rather than part of the core Bazel distribution.

Please send comments and suggestions to aiuto@google.com and bazel-dev@google.com, or respond to the issue/7444
We welcome discussion about capabilities and features
.

Table of Contents

Approvals

Name	Role	Date

Changelog

Editor	Comments	Date
aiuto	First draft, internal only	12 Nov 2018
aiuto	Incorporate android specific needs	27 Nov 2018
aiuto	reorder some requirements	2 Jan 2019
aiuto	make it clear that packages can have different rules from individual binaries	22 Feb 2019

References	### Design Options Design Options Proposal 1: <go/link> ### Links Project context: Other:

Objective

Provide a single license compliance mechanism that can be used at build time to enforce applicable constraints for all distributable binaries.

Background

Roughly.

- The code built into Blaze/Bazel today is insufficient for any complete solution.
- Various end user application teams have built ad-hoc solutions to meet their needs, but those solutions are inflexible and are cargo-cult re-adopted by other application teams.
- .

Goals

- Pull license compliance checking out of Blaze/Bazel and move it to Starlark rules. This has 2 benefits:
    - it will reduce the cost of adding new constraints going forward
    - it allows us to reuse the logic with Bazel, enabling its users to express the license constraints using a more powerful framework than currently exists
- Make it possible to provide different sets of license constraints for different types of targets. For example, a consumer phone application may not be allowed to include any LGPLed code, while a desktop application might.
- Provide tooling so that mobile and desktop applications can trivially include license notifications required for product shipment. This will allow all CLs impacting license gathering to be reviewed by legal, regardless of team.
- Provide a template for solving a class of similar problems. For example, restricting high value IP from accidentally shipping in mobile apps.

Non-goals

- To categorize licences used by third party code by scanning their text.
- To extract copyright notices from third party code.
- To define once-and-for-all the rules governing what licenses may be used in what combinations for what kinds of applications. The tools must be flexible enough so that a variety of schemes can be built.
- To automatically apply licenses based on package paths. E.g. “third_party” is not special unless *you* want it to be.

Glossary of terms

TBD: Brief summary of acronyms and domain specific concepts for the general reader.

Functional requirements

Capture features of licenses

FR: Specify license text. Capture the file holding the license text as provided by the author.
FR: Specify the license "type". The license "type" is a string which has a meaning to an organization's compliance department. It could be as simple as none|notice|restricted or as complex as a labeling of dozens of different types.
FR: Specify one-line copyright notice. Many applications need a consolidated list of the one line copyright notices. (E.g. "Copyright © YoYoDyne, Inc.")
FR: Specify the copyright holder name. Many applications list their third party software by name. It must be trivial for applications to gather the names of the copyright holders alongside the license text.

Automatic enforcement

FR: Binaries that are out of compliance should be detected at build time. bazel build or test should fail when trying to build a binary with non-compliant license mixes.
FR: Rules that build packages containing multiple binaries must be able to to mix components of different licenses according to rules appropriate for that package.
For example, a docker image might be allowed to contain a cc_binary built with all internally owned code alongside a cc_binary built from code under a specific license, as long as each is individually consistant.
FR: There can be an allowlist of targets which are granted exceptions to the normal rules. This gives us flexibility to take special cases under legal review and allow them on a case by case basis.

Usable from Blaze

FR: Full text of all licences as a "resource" target. Mobile applications should be able to point to a result of license checking to get the consolidated text of all licenses usable directly as a resource to be bundled into the application. No product team should have to build ad-hoc solutions to do this.
FR: rules like `android_application` should automatically have license resources included.
FR: There must be an ability to plug text manipulation hooks into license resource generation. For example, on android, license texts must be sorted together so that gzip compression works well.
FR: Rules that aren't buildable units should not have licenses applied. e.g., config_setting should not

have a license applied.

Miscellaneous

FR: Different targets in a single package must be able to be under different licenses. Some code in third_party brings in copies of other libraries. We need to be able to capture the set of licenses used rather than only that of the package top.

Reporting

FR: It must be trivial to gather a list of all the licensed software used in a binary. This makes human readable audits easier.
FR: Full text of all licenses as text/html. Same as above, but for non-mobile applications.
FR: Gather all copyright notices into a single block of text. Similar to the above, but for the one-line copyright notices.

Non-functional requirements

Flexible policy

FR: License types are not baked into Bazel/Blaze. The list of license types must be able to change over time without requiring new Blaze binary releases.
FR: Different classes of applications must be able to have different license constraints. The rules for end user applications might be different than those for packages used in serving infrastrurcture

Ease of use

FR: Adding license checking to an application should take no more than one (or two) additional rule(s) per top level object.
But, for the most part, license checking will be a default feature of rules that distribute objects. For example, rpm files or docker images.
FR: It must be possible to slow migrate from the existing license mechanisms. For Google, we can not have a flag day change. For Bazel, we can release the framework as soon as it is ready, and slowly migrate BUILD files in the wild to the new scheme.

Open source compatible

FR: We must be able to release the framework for license checking rules as open source. One of the important use cases for Bazel is one can audit the provenance of everything going into their application. Having a license checking capability is a key feature for customers in industries with strong audit and compliance requirements.