Bazel Catalog Design Proposal

Author: Xiaoyi Shi <ashi009@gmail.com>
Status: Draft
Created: 2022-02-14
Updated: 2022-02-15

Objective

Build a catalog for users with different levels of familiarity with Bazel.

User Groups

Beginners

They are unfamiliar with bazel and have little knowledge of how bazel works.
They want tutorials and guidelines.

- See what can be done with bazel (without purpose)
    - recommended rules (say official rules)
- Docs/codelabs
    - how to use bazel (build/test)
    - how to use a rule
- Tools
    - bazelisk
    - ibazel
    - gazelle

Intermediate Users

They know how to bazel build/test, edit build files, has some knowledge of how bazel works.
They want to explore what can be done with bazel, and migrate the existing codebase to bazel.

- Find rules for doing certain things (with purpose)
    - query by language
    - query on input file type
    - compare ruleset based on some metrics (eg. project health, traffic, etc.)
- Doc/codelabs
    - bazel internals
    - bazel query
    - how to write a rule
    - setting up remote cache

        - setting up remote execution
        - migration (xcode/maven/npm/…)
- Tools
    - tulsi

Experts

They are experts in bazel, knowing advanced concepts, internals, and usage of advanced bazel subcommands.

As the name implies, they are likely to be the decision maker for an organization on which ruleset/tool to use. So they want to know ruleset/tool in depth and they also so want to make upgrades to be smooth.

- Show rules in depth
    - toolchain
    - platform
    - configuration/transition
    - providers
    - remote execution capability
    - compatibility with other rules
    - design decisions
    - future plan
    - dev tool integration (eg. gopackagedriver)
    - CI results
    - vulnerabilities
- Docs
    - Release notes (changes since X, migration path, etc)
- Tools
    - ???

Data Expectation

Ruleset

The following table contains almost all the data relevant to a ruleset. The data has been categorized to meet the expectations of different user groups.

The background color identifies the difficulty of getting the data, and it increases in the order of green, yellow, orange, and red.

Type	User Groups	Source	Comment

Metadata - canonical name - repo - desc - authors - Semver guarantee, is it 1.0 yet? - releases (tags as a fallback) - bzlmod? compatibility - license	All	Automated from BCR/Repo Manually from rule authors	Either from MODULE.bazel file or inferred from the github repo metadata. (We may ask maintainers to add tags to their GitHub project)
Editorial Recommendations	Beginner	Manually from editors	With some structured formats? or just text?
Project Popularity	Beginner, Intermediate Users	Automated from Repo (release downloads, traffic)	https://shields.io/category/downloads conceptually identical to what npm is doing Searching for and choosing packages to download | npm Docs + starts/forks
Project Health	Beginner, Intermediate Users	Automated from Repo (questionable signal quality)	GitHub - dogweather/repo-health-check: Analyze a project: How are the maintainers doing? not sure if their evaluation is good, as the project itself is 8 years old.
Project Quality	Beginner, Intermediate Users	Automated from Repo (questionable signal quality)	Community metrics - GitHub Docs identical to npm's quality metric.
Project Activity	All	Automated from Repo (PRs/Issues/Commits)	https://shields.io/category/activity
Module Functionality - rules - providers - functions - aspects	All	Automated from Repo	Use stardoc to convert them to proto?
Module Quality - skylint - feature-related quality aspects (eg. is this python rule comes with a	All	Automated from Repo Manually from editors	The manual part is difficult to implement, as different rules need to be evaluated with different criteria. But this info could save users lots of time and help them make the decision.

hermetic toolchain? or is this proto rule force user to copy-paste a fixed set of deps?)			
Toolchain	Experts	Automated from Repo	easy for bzlmod, not sure how to do with
Platform	Experts	Automated from Repo	easy for bzlmod
Configuration/Transition	Experts	Automated from Repo	
RBE Support	Experts	Manually from rule authors	
Design Docs - Design Decisions - Future Plan	Experts	Manually from rule authors	
Dev Tool Integration - gazelle - LSP/IDE	Intermediate Users, Experts	Manually from rule authors	Maybe some predefined tags?
CI status of HEAD	Experts	Automated from Repo	reuse .presubmit.yml and run checks as github actions on releases? Also mentioned at https://github.com/bazel-contrib/SIG-rules-authors/issues/2#issuecomment-1142625002
Vulnerabilities	Experts	Manual?	Although a defect in a ruleset most likely affects only the build time. However, given the emergence of image rules, it may have far-reaching consequences. No source for this.

Other Resources

Other resources are less structured, so links to resources and editorial comments would be sufficient. However, if we want to make them easy to browse, these links need to be put into categories. eg. https://github.com/jin/awesome-bazel#toolchains.

Given that our focus is mostly on rulesets, this part can be ignored for now.

Data Ingest Pipeline

Requirements

To support client search/filter and avoid exhausting the GH API quota (even though most of the data is publicly readable), we need to materialize all the data to storage beforehand.

To make this work as easy as possible, instead of having everything built into a comprehensive service, we should build tools that dump data to storage as cronjobs. Once the data is ready, we can use an SSR framework to render the final catalog pages as cronjobs as well.

Process

1. Create a Github App to read automated signals

   Compared to using a pool of private tokens, this allows us to have independent quotas for each installation (5,000 requests per hour). Which will make it less likely to hit any quota limit.

2. Install the Github App to a ruleset repo that wants to be listed on the catalog

   This also verifies that whoever is asking to enroll is the maintainer of the ruleset, and thus has consented.

3. The Github App creates a commit/PR for adding a single ruleset once the installation is done

   This will add a directory (slightly different from today's big json file) and a metadata file for the ruleset. The reason for this is that we can throw additional data into that directory at later stages without worrying about merge conflicts.

4. Run cronjobs to retrieve automated signals

   Cronjobs are run with the Github App's identity and all the fetched data are stored in their dedicated data file (ideally as time-series.)

5. Run a tool to generate the catalog page as a cronjob and/or triggered by push events

   Could be a `bazel build` and then commit the generated files to gh-pages branch.

NOTE: manual signals can be added at any time with PRs
NOTE: use a pin/marker file to make the cronjob stateless
TODO: what to do when someone uninstalls??

TODO: figure out if https://github.com/just-the-docs/just-the-docs is good enough for the job