[Public] Roadmap Proposal: Bazel + Python

tl;dr This is an external-facing roadmap proposal for improvements to Bazel to make the tool useful to Python users.

Last updated: 2018-02-06
Authors: davidstanke@google.com, stewartr@google.com

Roadmap

Phase 1

Objective: Bazel supports standard Python build use cases (“Bazel works for Python developers”)

Bazel can build executables from pure Python (2 or 3) code on Linux, macOS and Windows

- Feedback:
  - the current rules are buggy, especially regarding some Python 3 assumptions
    - Various bazel tooling expects to run with a python 2 interpreter, but sometimes decides to use whatever py_runtime has been given. This breaks if said runtime is python 3.
    - Sometimes you **do** need to run tooling under the appropriate runtime (such as when fetching packages with pip, since the wheel you get is dependent upon the python version you are using). This doesn't happen in the right places, which is the crux of https://github.com/bazelbuild/rules_python/issues/37
  - Bazel users would like to ship the toolchain through Bazel (i.e., bundle a non-system interpreter)
  - Not realistic for everything to be rooted at tree root. Rooting issue seems to be causing issues where bazel run != bazel test.
  - Testing?
- Proposal:
  - ...

Bazel can build executables with C extension modules on Linux and macOS

- Feedback: Bazel can’t build C extension modules. Many third party libraries use C extension modules and can’t be built without this support.
- Proposal:
  - Operation Purple Boa
- See: https://groups.google.com/forum/#!topic/bazel-sig-python/_PfMBUkwulc

Bazel allows PyPI dependencies to be included in a build

- Feedback: the current rules are buggy
- Issues:
  - Issue
  - Issue
  - Issue: dealing with “extra”
  - Issue: hard-coded pygen, doesn’t use pyrun
  - Issue: doesn’t use cache (but is cache hermetic?)
  - Issue: needs to do a better job of showing progress (since these can take many minutes)
- Proposals:
  - dududko@gmail.com proposal: https://github.com/bazelbuild/rules_python/pull/61
  - Python builds should happen in a “clean” environment: dependencies will need to be declared explicitly; system-installed libraries should not be importable into Bazel builds
  - Bazel supports requirements.txt
  - Ideally, we should support three use cases:
    i. Run the tests
    ii. Run a binary locally
    iii. Build an artifact and throw it to the deployment system
  - See: https://groups.google.com/forum/#!topic/bazel-sig-python/_LzsWf8vxak

Bazel output can target Docker containers

- Feedback:
  - The current rules are buggy (Python 3 bug)
  - Users would like to build a Docker image from macOS
  - Bazel can output to a Docker container
    - Request: an easier way to add a py_library to a container
- Proposal:
  - ...

Bazel Python support is moved out of Bazel core and into rules_python

- Feedback: Bazel Python users should not have to investigate Bazel’s core Java code to understand how rules are working
- Response: It is Google’s intent to have language-specific rules live in a language-specific repository
- Proposals:
  - all new/updated rules will live in rules_python
  - existing rules will be migrated to rules_python over time

Bazel documentation is improved for Python

- Feedback:
  - It’s not immediately obvious that Bazel supports Python
  - Please show how to include a PyPI dependency in a build
- Proposals:
  - Bazel.build doesn’t highlights Python support and features Python-specific documentation
  - An example repository showcases pure Python and C extension module examples

Phase 2

Objective: Bazel supports common Python ecosystem tools (“Bazel is nice to use for Python developers”)

Key results

- Features
    - Bazel builds can use and include non-host Python interpreters
    - Bazel can package for, and publish to, PyPI
    - Bazel can output multiple Python binaries (e.g., by specifying python2 and python3 in something like py_runtime())
    - Bazel produces manylinux{1,2} binaries, Windows binaries, and macOS binaries
    - Bazel can build CLIF, Protocol Buffers, gRPC, Google Cloud Client Libraries (some of these potentially after phase 2)
- Integrations
    - Bazel supports the following Python static analysis tools:
        - Coverage
        - Pytest
            - Bonus for user-configurable test runner (as input to build)
            - Bonus for support of pytest’s coverage plugin.
        - Pyflakes
        - Pylint
        - Pycodestyle
        - Pydocstyle
        - Pytype (potentially phase 3)
        - mypy

For additional consideration

The list of objectives beyond phase 2 is still to be defined. Items under consideration include:

- Feature parity for Python developers who use Windows
- Hermetic PAR files
    - Q: is this solved via non-host Python interpreters? Or is there additional work?
- Better cross-compilation support
- Make native py_library rules accessible from Skylark (evan.jones@bluecore.com)
    - Notably, the import attribute is not accessible. This would make it possible for Skylark rules to depend on py_library "correctly" (maybe). This might also be completely useless, if the "path forward" is "get rid of the native py_library rules". However, I suspect there may be a very long transition period where both sets of rules exist, so this would make it possible to be compatible: https://github.com/bazelbuild/bazel/issues/2617
- Bazel REPL mode
    - “I want the same as pants repl or sbt console or lein repl or stack ghci which is a language level repl for interacting with the code you have written.” (this is ~attaching stdin to a bazel run process)
    - PEX supports "interpreter" binaries which act like a Python interpreter with the given dependencies: e.g. with no args: interactive prompt; with an arg: run that script. Its very useful to provide a "packaged environment" that can be used with arbitrary scripts, or poked at for debugging. https://pex.readthedocs.io/en/stable/buildingpex.html#specifying-entry-points

- IDE integration, e.g. PyCharm