

DebianでBazelを 利用したビルドを 楽にしたい

dh-bazelというdebhelperによる汎用的な解決策の提案

Kentaro Hayashi

ClearCode Inc.

2026年11月 東京エリア・関西合同Debian勉強会（オンライン開催）





スライドは Rabbit Slide Showにて公開済み

- DebianでBazelを利用したビルドを楽しみたい
 - <https://slide.rabbit-shocker.org/authors/kenhys/tokyodebian-dh-bazel-202611>





今日の発表のまとめ

- dh-bazelを開発しました 🍀
 - Bazelに対応したdebhelperのbuildsystem
 - 従来の課題: Bazelを採用したソフトウェアをDebianパッケージ化する際、既存のdebhelperのビルドワークフローに乗せにくい
 - 解決手法: dhのbuildsystemへBazelを統合し、dhを活用できるようにした
 - **メリット: Bazelを採用したパッケージをDebianに投入しやすくなる(かもしれない)**



今日の発表内容

- Bazelとは？
- DebianにおけるBazelの利用状況
- なぜBazel対応が必要なのか？
- Bazelをdebhelperにどう組み込むか



Bazelとは？

- Googleが開発したオープンソースのビルド・テストツール
- 巨大なコードベースを持つプロジェクトで採用されている/いた
 - TensorFlow
 - Kubernetes(Bazel廃止済み)
- **再現性を重視した**ビルドツール
 - 必要な依存ライブラリー等をダウンロードしてビルドする



DebianにおけるBazelの利用状況

- 採用パッケージはbazel-*系を除くとかなり限定的(codesearch.d.n調べ)
 - mozc (3.33.6133+ds1-0.1から)
 - android-cuttlefish (仮想Androidデバイス AVD)
- Bazelが問題になっている事例
 - tensorflowはfull版ではなく lite版(CMake)を採用



Bazel対応がDebianですすまないのはなぜか？

- Bazel 7.7.1がDebian unstableで利用可能になったのは最近(2026/6月末頃)
 - 従来Bazel (bazel-bootstrap)は4.2.3までしかなかった
 - より新しいBazelを必要とするソフトウェアはパッケージ化できない 🤔



今日の発表の内容

- Bazelとは？
- DebianにおけるBazelの状況
- **なぜBazel対応が必要なのか？**
- Bazelをdebhelperにどう組み込むか



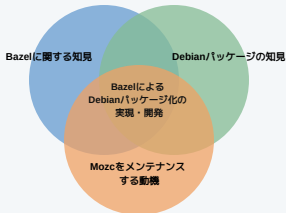
なぜBazel対応が必要なのか？

- より新しいMozcをDebianで採用できなかった
 - MozcはGYPからBazelに移行済みでGYP維持することに先はない
 - Bazelに追従できない場合、あえてGYPにとどまるか、2.29.5160.102に塩漬ける未来
 - どちらもメンテナンスコスト増大を招く。誰がそのコストを払えるのか？



DebianでMozcをBazel対応させる難しさ

Mozcをメンテナンスするのに必要なこと





DebianでBazelを利用する場合の課題

- Bazelは(基本的に)ビルド中のネットワークアクセス前提
 - **debian/パッケージのビルドでは禁止**
- Bazelのオフライン化はDebianポリシーになじまない
 - 依存関係を事前にまるっと抱え込むということ
 - いわゆるvendorライブラリーを使う方向性
 - **debianでは推奨されない**
- 結局どうするのか？



Debianでは無効化ハックが必要

■ 依存情報の解決

- Bazel Central Registry(<https://registry.bazel.build/>)にアクセスさせない
- 使わないモジュールは無効化する

■ 依存ライブラリーの解決

- システムライブラリーをリンクするように改変



参考: Bazelの基本 (C/C++想定)

・モジュールの定義

- MODULE.bazel

- 該当モジュールが何に依存しているのかを明示する

- BUILD.bazel

- 該当モジュールをどうやってビルドするのかを明示する

・**理屈の上では、モジュールを置き換えてしまえばビルドできるはず**



参考: 他のディストリビューションの事例

- Fedora

- Mozc 2.29.5111.102
 - GYPパッチを当てて維持

- openSUSE

- Mozc 3.34.6239
 - bcr.tar.gz, dependencies.tarといった依存関係のメタデータを事前に抱え込みビルド時に静的リンク

- (Debianの)システムライブラリーをリンクさせる戦い方とは方針が異なる



Mozcの事例 (指定しているオプションの抜粋)

```
# Override module and repository to disable network access completely
BAZEL_OPTIONS= {
  -arc4random_jc {
    -colorno {
    -compilation_modeopt {
    -configure_linux {
    -cursesno {
    -features-compiler_param_file {
    -lockfile_modernopt {
    -mutable_workspaces {
    -override_modulebabel-cpp:./debian/third_party/babel-cpp {
    -override_moduleapple_support:./debian/third_party/apple_support {
    -override_modulebuildzmq:./debian/third_party/buildzmq {
    -override_modulebazel:./debian/third_party/bazel {
    -override_modulegoogletest:./debian/third_party/gtest {
    -override_moduleplatform:./debian/third_party/platform {
    -override_moduleprotobuf:./debian/third_party/protobuf {
    -override_modulemodules_android:./debian/third_party/rules_android {
    -override_modulemodules_android_ndk:./debian/third_party/rules_android_ndk {
    -override_modulemodules_apple:./debian/third_party/rules_apple {
    -override_modulemodules_gcc:./debian/third_party/rules_gcc {
    -override_modulemodules_pkgconfig:./debian/third_party/rules_pkg {
    -override_modulemodules:./debian/third_party/tls {
    -override_repository_main-repo_rules-ja_usage_dict:third_party/usage_dict {
    -override_repository_main-repo_rules-material_icon_dictionary_png:third_party/material_icon_dictionary_png {
    -override_repository_main-repo_rules-material_icon_dictionary_svg:third_party/material_icon_dictionary_svg {
    -override_repository_main-repo_rules-material_icon_properties_png:third_party/material_icon_properties_png {
    -override_repository_main-repo_rules-material_icon_properties_svg:third_party/material_icon_properties_svg {
    -override_repository_main-repo_rules-material_icon_tool_png:third_party/material_icon_tool_png {
    -override_repository_main-repo_rules-material_icon_tool_svg:third_party/material_icon_tool_svg {
    -override_repository_main-repo_rules-dip_code_jigyo:third_party/dip_code_jigyo {
    -override_repository_main-repo_rules-dip_code_kan_all:third_party/dip_code_kan_all {
    -repo_archBAZEL_CXXOPTS=stdc++20 {
    -repository_disable_download {
    -sandbox_add_mount_pair:usr {
    -stripmev {
    -subcommands
```



Q. 依存関係を一部無効化する方法

- A. 使っていないモジュールは
--override_moduleで何もしないBUILD.bazelとMODULE.bazelを参照させる
 - e.g. --override_module=apple_support=../debian/third_party/apple_support

```
../debian/third_party/apple_support/BUILD.bazel は空のファイル  
> cat ../debian/third_party/apple_support/MODULE.bazel  
module(  
  name = "apple_support",  
  version = "system",  
)
```



Q. システムのライブラリーを使わせる方法

A. bazel実行時に`--override_module`でシステムライブラリーを参照させるようにする

- e.g. `--override_module=zlib=../debian/third_party/zlib`

```
> cat debian/third_party/zlib/BUILD.bazel
package(default_visibility = ["//visibility:public"])

cc_library(
    name = "zlib",
    linkopts = ["-lz"],
)
> cat debian/third_party/zlib/MODULE.bazel
module(
    name = "zlib",
    version = "system",
)
```



Bazel利用時の面倒くささ

- Bazelの起動オプションと、ビルドオプションの調整
- パッケージ固有の依存関係の調整
 - `--override_module`で使わないモジュールは無効化
 - `--override_module`でシステムライブラリーをリンク
- 前者に関して、各パッケージでばらばらに同じようなことをやるのは無駄では？ 😞



Bazel利用時の ビルドシステムを改善したい

- そもそもdebhelperのサポートがないのがいけない
 - なぜbazelコマンドをdebian/rulesで直打ち実行しないといけないのか 🤔



debhelperでサポートする方法

debhelperを拡張する方法

■ buildsystem

- dh_auto_*が裏で実行するコマンドを制御する
- --buildsystem=で指定するもの (e.g. cmake,meson,pybuild)

■ addon

- ビルド途中に追加のdh_*フックを割り込ませる (e.g. autoreconf)
- dh-sequence-xxxをBuild-Dependsに追加して有効化
- 昔はdh --with=で指定していたやつ



debhelperでサポートする方法

debhelperを拡張する方法

■ buildsystem

- dh_auto_*が裏で実行するコマンドを制御する
- --buildsystem=で指定するもの (e.g. cmake,meson,pybuild)
- **bazelでサポートすべきなのはこちら**

■ addon

- ビルド途中に追加のdh_*後処理を割り込ませる
- dh-sequence-xxxをBuild-Dependsに追加して有効化
- 昔はdh --with=で指定していたやつ



Debianのbuildsystemのしくみ

- 実装言語はPerl
- `/usr/share/perl5/Debian/Debhelper/Buildsystem/`にモジュールを配置する
 - Bazelサポートなら`bazel.pm`を配置すればよい
- `sub configure`, `sub build`, `sub clean`等を実装する
 - `--buildsystem=bazel`を指定すると`dh_auto_*`が上記を呼び出す



dh-bazelのコンセプト

- debhelperのbuildsystemの1つとしてBazel向けに提供
- bazelコマンドをあまり意識しないで使えること
- よくある起動オプションやオプションは指定しなくてもいい感じに動作すること



最終的な目標

- debian/rulesを簡潔に記述できること!

```
#!/usr/bin/make -f
# -*- makefile -*-
#

%:
    dh $@ --buildsystem=bazel

override_dh_auto_build:
    dh_auto_build -- //:hello
```



dh-bazelのコンセプトを実現するために

- 起動オプションとビルドオプションを整理
- 3つの機能を搭載
 - ダミーモジュール対応
 - pkgconf対応
 - Build-Dependsのfallback対応



dh-bazel ダミーモジュール対応

- ・システムライブラリーを参照しやすいように定義ファイルを用意
 - debian/rulesでexport DH_BAZEL_OVERRIDE_MODULES=zlibするだけでいい
 - dh-bazelのダミーモジュールを定義したファイル (MODULE.bazel,BUILD.bazel)を参照して勝手に解決
 - 裏で--override_module=...定義に自動変換してbazelを実行する
 - あらゆるライブラリーに対応するにはコストがかかりすぎる欠点



dh-bazel pkgconf対応

- ・pkgconfを利用するパッケージ向けにdh-bazelが変換
 - debian/rulesでexport DH_BAZEL_PKGCONF_MODULES=lz4=liblz4するだけでいい
 - lz4=liblz4は必要なモジュールがlz4で、pkgconfがliblz4.pcを探すという意味
 - dh-bazelのダミーモジュールを定義したファイル (MODULE.bazel,BUILD.bazel)を自動生成して勝手に解決
 - 裏で--override_module=...定義に自動変換してbazelを実行する
 - C/C++に限定されるがより汎用的に対処可能



dh-bazel

Build-Dependsのfallback対応

- dh-bazelがBuild-Depends:をみて自動変換
 - DH_BAZEL_OVERRIDE_MODULESやDH_BAZEL_PKGCONF_MODULESが空振りしたときの最終手段
 - debian/controlのBuild-Dependsから-devパッケージを探して、それに紐づいている.pcからリンク対象等を見つけて勝手に解決(ただし精度は低いし、誤爆もすることもある)
 - 裏で--override_module=...定義に自動変換してbazelを実行する



現在の開発状況

■ <https://salsa.debian.org/bazel-team/dh-bazel>

- 元々は個人リポジトリで開発していたのをbazel-teamへ移行済み
- 依存しているbazel-buildtoolsをITP, dfsg-teamのレビュー待ち
 - <https://bugs.debian.org/1151519>
- PoCのため、まだDebian unstableにはアップロードしていない



DEMO

- `DH_BAZEL_OVERRIDE_MODULES`
- `DH_BAZEL_PKGCONF_MODULES`
- `fallback`



今日の発表のまとめ

- dh-bazelを開発しました 🍡
 - Bazelに対応したdebhelperのbuildsystem
 - 従来の課題: Bazelを採用したソフトウェアをDebianパッケージ化する際、既存のdebhelperのビルドワークフローに乗せにくい
 - 解決手法: dhのbuildsystemへBazelを統合し、既存のワークフローを活用できるようにした
 - Bazelを採用したパッケージをDebianに投入しやすくなるメリット
 - 次のステップ: dh-bazelをDebian入りさせてMozcをdh-bazelを使ってビルドできるようにする