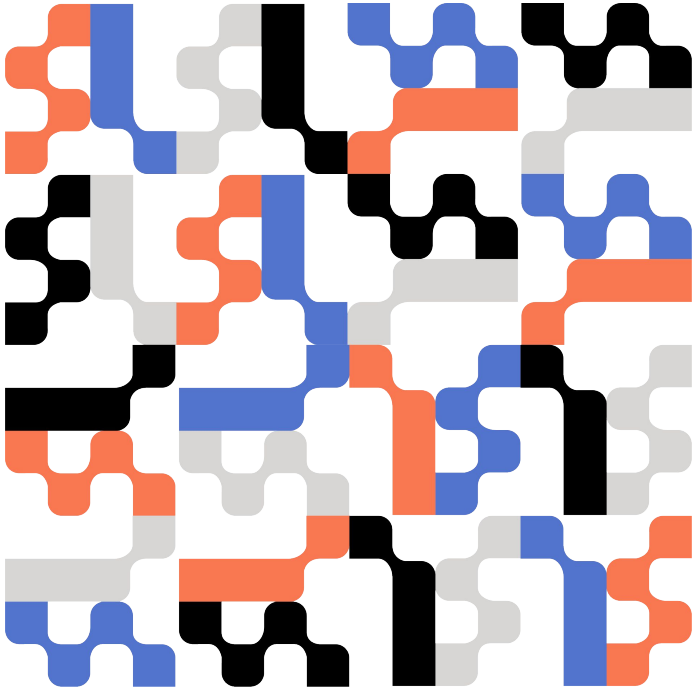
MODUS CREATE | TWEAG by Modus Create

BAZEL BUILDS & BUILDS WITH BAZEL

Packaging Bazel and Bazel-based packages with Nix



Guillaume Maudoux – 04.Feb.2024

GUILLAUME MAUDOUX

+ Computer scientist
    + Build systems enthusiast
        + Consultant in **Nix** and **Bazel**
        + *but my favorite build system remains Tup ;-)*


@layus


blog.layus.be

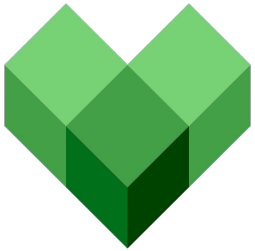

www.tweag.io/blog/

And I packaged Bazel 7.0.0 for nixpkgs 🎉

Work funded by Intuitive Surgical and Tweag

BUILD SYSTEMS CONFLICTS

BAZEL		NIX
File based	🤝	**Package** based
Uses local and remote caches	😕	Sandboxes execution
Uses remote execution	😕	Sandboxes execution
Assumes FHS everywhere	😬	Not FHS compliant
Loves precompiled dependencies	😭	Does not run precompiled binaries
Downloads its own dependencies	🤯	Sandboxes execution



BAZEL IS HARD TO PACKAGE IN NIX

Bazel hardcodes /usr/bin everywhere

```
  -e 's!/usr/local/bin/bash!${bashWithDefaultShellUtils}/bin/bash!g' \
  -e 's!/usr/bin/bash!${bashWithDefaultShellUtils}/bin/bash!g' \
  -e 's!/bin/bash!${bashWithDefaultShellUtils}/bin/bash!g' \
  -e 's!/usr/bin/env bash!${bashWithDefaultShellUtils}/bin/bash!g' \
  -e 's!/usr/bin/env python2!${python3}/bin/python!g' \
  -e 's!/usr/bin/env python!${python3}/bin/python!g' \
  -e 's!/usr/bin/env!${coreutils}/bin/env!g' \
  -e 's!/bin/true!${coreutils}/bin/true!g'
```

Bazel prevents system sleep

```
-    BAZEL_CHECK_EQ(g_sleep_state_assertion, kIOPMNullAssertionID);
-    CFStringRef reasonForActivity = CFSTR("build.bazel");
-    IOReturn success = IOPMAssertionCreateWithName(
-        kIOPMAssertionTypeNoIdleSleep, kIOPMAssertionLevelOn, reasonForActivity,
-        &g_sleep_state_assertion);
-    BAZEL_CHECK_EQ(success, kIOReturnSuccess);
-  }
-  g_sleep_state_stack += 1;
-  return 0;
+  // Unreliable, disable for now
+  return -1;
 }
```

Bazel appends an empty arg for gcc (?)

```
 # Call the C++ compiler
-%{cc} "$@"
+if [[ ${*: -1} = "" ]]; then
+    %{cc} "${@:0:$#}"
+else
+    %{cc} "$@"
+fi
```

Requires patches in nested archives

```
# unzip builtins_bzl.zip so the contents get patched
builtins_bzl=src/main/java/com/google/devtools/build/lib
unzip ''${builtins_bzl}.zip -d ''${builtins_bzl}_zip >/d
rm ''${builtins_bzl}.zip
builtins_bzl=''${builtins_bzl}_zip/builtins_bzl
```

SELECT ISSUES & SOLUTIONS

Package Bazel

1. The JAVA toolchain
2. Setting the right $PATH
3. Fetching dependencies of the build

Package projects built with Bazel

4. Picking the right bazel version
5. The JAVA toolchain, again

JAVA TOOLCHAIN

JAVA TOOLCHAIN & PREBUILT BINARIES

Bazel hardcodes prebuilt tools for most platforms

```
alias(
    name = "singlejar",
    actual = ":singlejar_prebuilt_or_cc_binary",
)

alias(
    name = "singlejar_prebuilt_or_cc_binary",
    actual = select({
        "@bazel_tools//src/conditions:darwin_arm64": ":prebuilt_singlejar_darwin_arm64",
        "@bazel_tools//src/conditions:darwin_x86_64": ":prebuilt_singlejar_darwin_x86_64",
        "@bazel_tools//src/conditions:linux_x86_64": ":prebuilt_singlejar_linux",
        "@bazel_tools//src/conditions:windows": ":prebuilt_singlejar_windows",
        "//conditions:default": "@remote_java_tools//:singlejar_cc_bin",
    }),
)
```

SOLUTION: CUSTOM JAVA TOOLCHAINS

Define and register nonprebuilt_toolchain_java21 that overrides all the prebuilt defaults

```
+    default_java_toolchain(
+        name = "nonprebuilt_toolchain_java" + str(version),
+        configuration = NONPREBUILT_TOOLCHAIN_CONFIGURATION,
+        java_runtime = "@local_jdk//:jdk",
+        source_version = str(version),
+        target_version = str(version),
```

```
using all the tools from sources.
NONPREBUILT_TOOLCHAIN_CONFIGURATION = dict(
    ijar = [Label("@remote_java_tools//:ijar_cc_binary")],
    singlejar = [Label("@remote_java_tools//:singlejar_cc_bin")],
    header_compiler_direct = [Label("@remote_java_tools//:TurbineDirect")],
)
```

```
# Default java_toolchain parameters
DEFAULT_TOOLCHAIN_CONFIGURATION = dict(
    java_runtime = Label("//toolchains:remotejdk_21"),
    singlejar = [Label("//toolchains:singlejar")],
```

CUSTOM TOOLCHAIN IS FRAGILE

It can easily get broken by upstream changes because that config is mostly untested

Turbine native image broke NONPREBUILT_TOOLCHAIN_CONFIGURATION #166

Closed 0 comments

layus commented 3 weeks ago

I believe `d5b3ecf` broke the NONPREE
NONPREBUILT_TOOLCHAIN_CONFIGU

```
header_compiler_direct = [Label("
```

1 toolchains/default_java_toolchain.bzl

```
@@ -130,6 +130,7 @@ PREBUILT_TOOLCHAIN_CONFIGURATION = dict(
130 130 NONPREBUILT_TOOLCHAIN_CONFIGURATION = dict(
131 131     ijar = [Label("@remote_java_tools//:ijar_cc_binary")],
132 132     singlejar = [Label("@remote_java_tools//:singlejar_cc_bin")],
    133     header_compiler_direct = [Label("@remote_java_tools//:TurbineDirect")],
133 134 )
134 135
135 136 _DEFAULT_SOURCE_VERSION = "8"
```

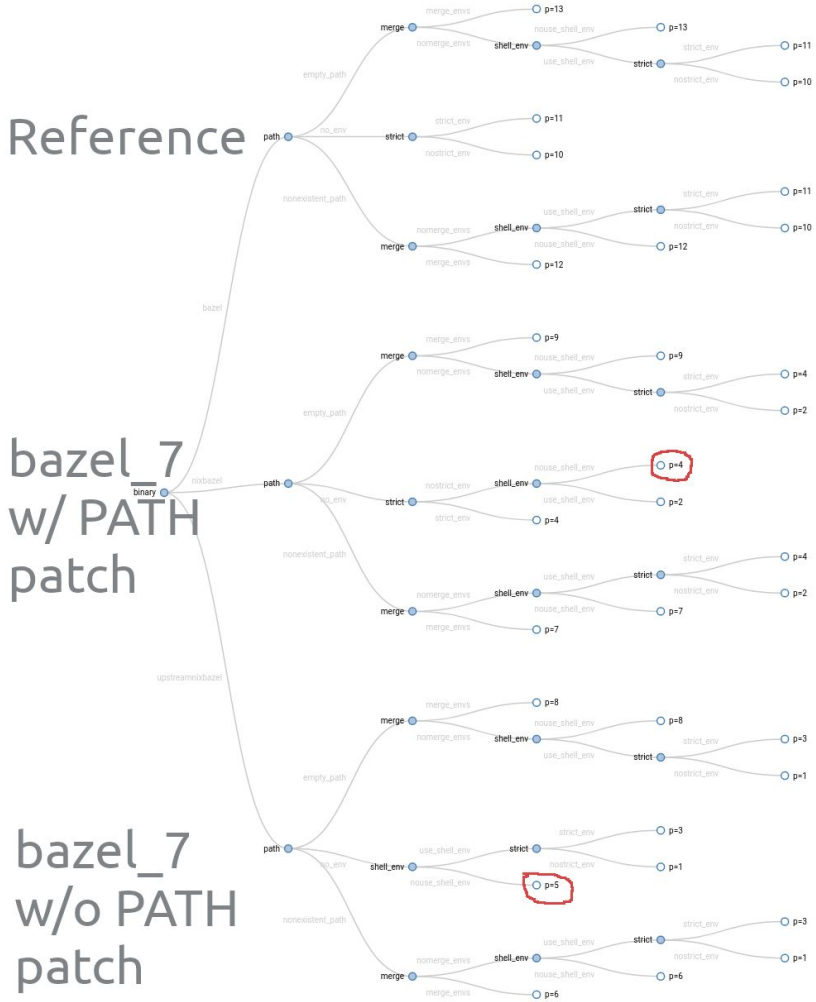
SETTING
THE RIGHT $PATH

DISCREPANCIES IN $PATH

Actions see PATH=/no-such-path

+ We have a fix for that in theory 🤔
+ Need to fix the fix

```
  bashWithDefaultShellUtilsSh = writeShellApplication {
    name = "bash";
-   runtimeInputs = defaultShellUtils;
    text = ''
      if [[ "$PATH" == "/no-such-path" ]]; then
        export PATH=${defaultShellPath}
```

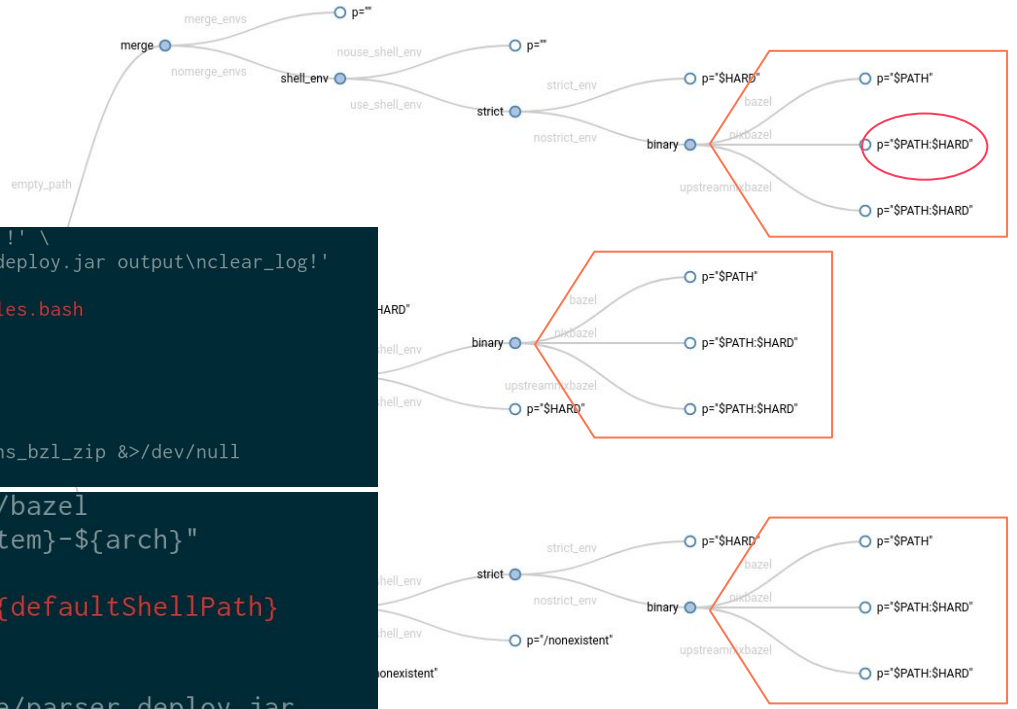


DISCREPANCIES IN $PATH

PATH is modified in too many places

```
  -e 's!bazel_build !bazel_build src/tools/execlog:parser_deploy.jar !' \
  -e 's!clear_log!cp $(get_bazel_bin_path)/src/tools/execlog/parser_deploy.jar output\nclear_log!'

# append the PATH with defaultShellPath in tools/bash/runfiles/runfiles.bash
echo "PATH=\$PATH:${defaultShellPath}" >> runfiles.bash.tmp
cat tools/bash/runfiles/runfiles.bash >> runfiles.bash.tmp
mv runfiles.bash.tmp tools/bash/runfiles/runfiles.bash

# reconstruct the now patched builtins_bzl.zip
pushd src/main/java/com/google/devtools/build/lib/bazel/rules/builtins_bzl_zip &>/dev/null
  zip ../builtins_bzl.zip $(find builtins_bzl -type f) >/dev/null
```

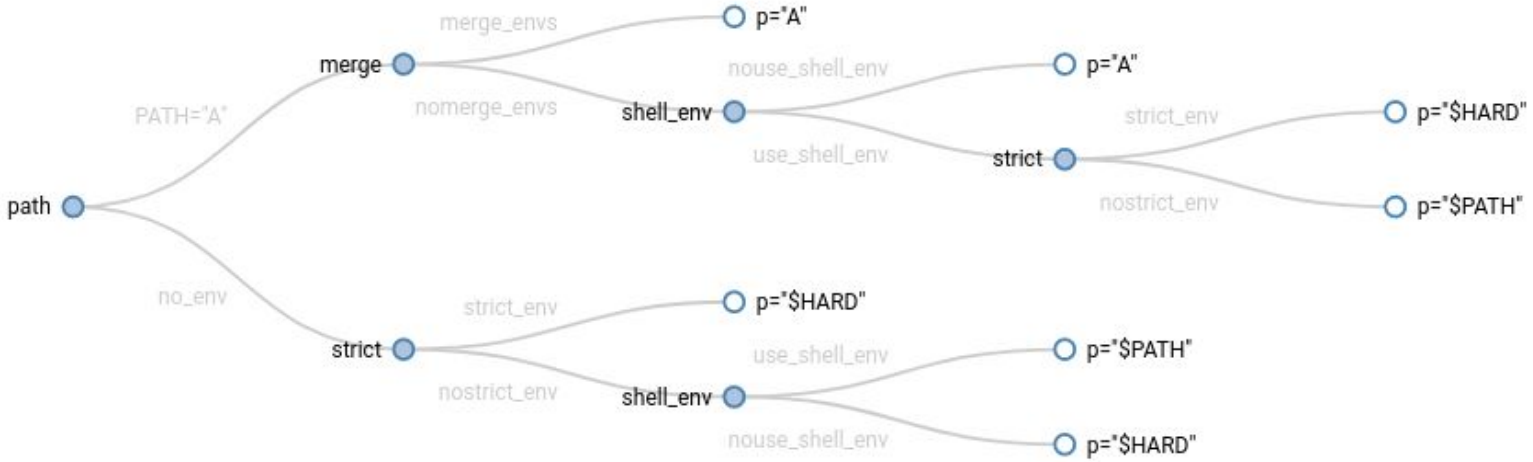
```
cp ./bazel_src/scripts/packages/bazel.sh $out/bin/bazel
versionned_bazel="$out/bin/bazel-${version}-${system}-${arch}"
mv ./bazel_src/output/bazel "$versionned_bazel"
wrapProgram "$versionned_bazel" --suffix PATH : ${defaultShellPath}

mkdir $out/share
cp ./bazel_src/output/parser_deploy.jar $out/share/parser_deploy.jar
```



MODUS CREATE | TWEAG by Modus Create

DISCREPANCIES IN $PATH

$HARD hardcoded path is "/bin:/usr/bin:/usr/local/bin" for upstream,
and ${lib.makeBinPath defaultShellUtils} for nix package



See --incompatible_strict_action_env

UPSTREAMING NIX SUPPORT IN BAZEL

So many patches is hard to maintain

+ Generally we need to configure the build
+ But Bazel prefers hardcoded select statements

  There is barely any configurePhase for Bazel.

+ Bazel needs more flags for configuration

  Bazel has support for configuration flags,

  but they are seldom used.

(PRE)FETCHING BUILD DEPENDENCIES

EXTRACTING URLS FROM WORKSPACE

Bazel syntax is a subset of Python

WORKSPACE (@ 2192a05)

```
162
163 http_archive(
164     name = "com_google_googletest",
165     sha256 = "81964fe578e9bd7c94dfdb09c8e4d6e6759e19967e397dbea48d1c10e45d0df2",
166     strip_prefix = "googletest-release-1.12.1",
167     urls = [
168         "https://mirror.bazel.build/github.com/google/googletest/archive/refs/tags/release-1.12.1.tar.gz",
169         "https://github.com/google/googletest/archive/refs/tags/release-1.12.1.tar.gz",
170     ],
171 )
172
```

EXTRACTING URLS FROM WORKSPACE

Bazel syntax is a subset of Python, so we can execute it like Python 🤓

```python
# just the kw args are the dict { name, sha256, urls … }
def http_archive(**kw):
    http_archives.append(kw)
# like http_file
def http_file(**kw):
    http_archives.append(kw)


# stubs for symbols we are not interested in
# might need to be expanded if new bazel releases add symbols to the workspace
def workspace(name): pass
def load(*args): pass
def bind(**kw): pass

# execute the WORKSPACE like it was python code in this module,
# using all the function stubs from above.
with open(sys.argv[1]) as f:
    exec(f.read())

# transform to a dict with the names as keys
d = { el['name']: el for el in http_archives }

print(json.dumps(d, sort_keys=True, indent=4))
```

WORKSPACE (@ 2192a05)

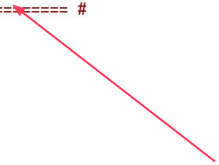
```
162
163 http_archive(
164     name = "com_google_googletest",
165     sha256 = "81964fe578e9bd7c94dfdb09c8e4d6e6759e19967e397dbea48d1c10e45d0df2",
166     strip_prefix = "googletest-release-1.12.1",
167     urls = [
168         "https://mirror.bazel.build/github.com/google/googletest/archive/refs/tags/release-1.12.1.tar.gz",
169         "https://github.com/google/googletest/archive/refs/tags/release-1.12.1.tar.gz",
170     ],
171 )
172
```

EXTRACTING URLS FROM WORKSPACE

Bazel syntax is a subset of Python, so we can execute it like Python 🤓
Enter bazelmod 😱

WORKSPACE

```
# ========================================== #
# All dependencies have been moved to MODULE.Bazel #
# ========================================== #
```



Not executable Python anymore

EXTRACTING URLS FROM MODULE.bazel.lock

The new lockfile format is pure json, easy to parse with...

+ jq
+ Python
+ Nix !

```
lockfile = builtins.fetchurl {
  url = "https://raw.githubusercontent.com/bazelbuild/bazel/release-${version}/MODULE.bazel.lock";
  sha256 = "sha256-5xPpCeWVKVp1s4RVce/GoW2+fH8vniz5G1MNI4uezpc=";
};

# Two-in-one format
distDir = repoCache;
repoCache = callPackage ./bazel-repository-cache.nix {
  inherit lockfile;
};
```

EXTRACTING URLS FROM MODULE.bazel.lock

The new lockfile format is pure json, easy to parse with Nix.

But

+ There are already 3 versions and v4 underway
+ Urls and hashes can be found anywhere
+ We ***need*** this to build Bazel **and** packages built with Bazel

PICK THE RIGHT BAZEL VERSION

BAZEL WANTS SPECIFIC VERSIONS

BAZEL

.bazelversion

```
1 7.0.2
```

+ Just ignore, take same major release
+ Used to work, but now FAILS

NIXPKGS

bazel_7

Build tool that builds code quickly and reliably

Name: `bazel` Version: **7.0.0** Homepage Source License: Apache License 2.0

bazel_5

Build tool that builds code quickly and reliably

Name: `bazel` Version: **5.4.1** Homepage Source License: Apache License 2.0

bazel_4

Build tool that builds code quickly and reliably

Name: `bazel` Version: **4.2.2** Homepage Source License: Apache License 2.0

bazel

Build tool that builds code quickly and reliably

Name: `bazel` Version: **6.4.0** Homepage Source License: Apache License 2.0

LOCKFILE IS COUPLED WITH BAZEL VERSION

Lockfile contains “builtin” dependencies for builtin Bazel rules
They change with Bazel version, independently of project config

Generated from project lockfile

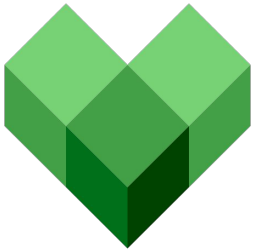
Generated from Bazel own sources lockfile

```
mergedDistDir = symlinkJoin {
  name = "mergedDistDir";
  paths = [ localDistDir distDir ];
};

testBazel = bazelTest {
  name = "bazel-test-cpp";
  inherit workspaceDir;
  bazelPkg = bazel;
  bazelScript = ''
    ${bazel}/bin/bazel build //... \
      --enable_bzlmod \
      --verbose_failures \
      --repository_cache=${mergedDistDir} \
```

THE JAVA TOOLCHAIN (AGAIN)

STRIVING FOR SEAMLESS USER EXPERIENCE

GOAL Everything should work by default in three main modes of nix operation
It should be easy to revert the introduced changes



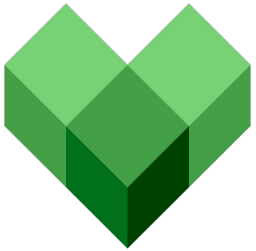
```
bazelRC = writeTextFile {
  name = "bazel-rc";
  text = ''
    startup --server_javabase=${runJdk}

    # Register nix-specific nonprebuilt java toolchains
    build --extra_toolchains=@bazel_tools//tools/jdk:all
    # and set bazel to use them by default
    build --tool_java_runtime_version=local_jdk
    build --java_runtime_version=local_jdk

    # load default location for the system wide configuration
    try-import /etc/bazel.bazelrc
  '';
};
```

/etc/bazel.bazelrc

```
1 <whatever>
```

STRIVING FOR SEAMLESS USER EXPERIENCE

GOAL Everything should work by default in three main modes of nix operation
It should be easy to revert the introduced changes



```
bazelRC = writeTextFile {
  name = "bazel-rc";
  text = ''
    startup --server_javabase=${runJdk}

    # Register nix-specific nonprebuilt java toolchains
    build --extra_toolchains=@bazel_tools//tools/jdk:all
    # and set bazel to use them by default
    build --tool_java_runtime_version=local_jdk
    build --java_runtime_version=local_jdk

    # load default location for the system wide configuration
    try-import /etc/bazel.bazelrc
  '';
};
```

Using flags allows easy override...

Precedence of --extra_toolchains toolchains is wrong #19945

Closed 3 comments

A fix for this issue has been included in Bazel 7.0.0 RC5.

/etc/bazel.bazelrc

```
1 <whatever>
```

UPSTREAMING SUPPORT FOR NIX BUILDS

- Less need as we can work around issues
- Avoid upstreaming nix-specific stuff
- Keep improving our tooling
  - Extract and merge lockfile deps
  - Sane defaults in all situations
  - Keep everything overridable

Not completely there yet...

- Cannot compile Bazel from real sources
- Not fully bootstrapped (uses only the ./bootstrap script)
- Still cumbersome to packages bazel packages

+ Upstream is keen to listen and merge
+ Many reviewers and testers on the PR
+ Improves with each iteration, on both sides

Special thanks to
@uri-canva
@boltzmannrain
@aaronmodal
@dmivankov
@folliehiyuki
@malt3
@name-snrl
@thimothyklim
@Strum355
@rickvanprim
Intuitive Surgical
Tweag colleagues
... and all of you !

I LOVE HARD CHALLENGES !
AND YOU ?

MODUS CREATE | TWEAG by Modus Create