What's Bazel? Why should you care?

What is this talk about?

- What is a build system?
- A short history of build systems
- Challenges faced by builds at a large scale
- Introduction to Bazel

Who am I?

Antonio Di Stefano

DevEx engineer @ Engflow

What is a build system?

A build system is a tool that helps you go from source code to deployable artifacts. It includes tasks such as:

- Compiling code
- Generating code
- Running tests
- Packaging artifacts (e.g., Docker images, JARs, .deb, .tar.gz, etc.)

Challenges of Build Systems

- Speed and Efficiency
- Reproducibility
- Configurability
- Extensibility
- Integrations

History of Build Systems

DIY builds with shell scripting

```
g++ -Wall ... -o lib lib.cc
g++ -Wall ... -o main main.cc
g++ -Wall ... -o test test.cc
./test
```

Pros:

- Maintaining this build will guarantuee job stability for a while
- Can technically be used with any lanuage/platform

Cons:

- Not very portable
- Requires setup for each workstation
- No caching from the build
- Hard to parallelize and speed-up
- Bespoke configuration
- No integration with tools
- Cannot trivially target a subset of the build

Make

```
CXX=g++
CXX_FLAGS=-Wall ...
lib:
  $(CXX) $(CXX_FLAGS) -o lib lib.cc
main: lib
  $(CXX) $(CXX_FLAGS) -o main main.cc
test: lib
  $(CXX) $(CXX_FLAGS) -o test test.cc
run_test:
  ./test
```

Pros:

- Is able to cache build targets
- Can parallelize build target (eg: `build_main` and `build_test/test`)
- Can also be used with any lanuage/platform

Cons:

- As portable as a shell script
- Naive caching based on timestamp
- Hard to reuse tooling
- Hard to read

CMake

```cmake
project(stuff)
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wall ...")

add_library(lib SHARED lib.cc)

add_executable(main main.cc)
target_link_libraries(main lib)

add_executable(test_bin test.cc)
target_link_libraries(test_bin lib)

add_custom_target(
  test
  COMMAND ./test_bin
  DEPENDS test_bin)
```

Pros:

- Reusable functions and macros
- Makes it easier to write portable build scripts
- Has good support for a bunch of compiled languages out-of-the-box
- Well integrated with 3rd party tooling
- Very configurable
- Someone wrote a raytracer with cmake: https://github.com/64/cmake-raytracer

Cons:

- Still suffers from the same issue as the underlying build system

Languge-specific build systems

Example: Maven

```xml
<project ... >
    <groupId>world.hello</groupId>
    <artifactId>hello-world</artifactId>
    <packaging>jar</packaging>
    <version>0.0.1</version>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>
</project>
```

Pros:

- Drop-in solution for a specific language
- Easy to use
- Usually has support for dependency management out-of-the-box
- Very well integrated within the ecosystem

Cons:

- Specific to a single language
- Hard to tune
- Hard to integrate different projects together

What is Bazel?

- Open-source build system developed by Google
- Designed for large, complex software projects
- Scalable, deterministic, and supports multiple languages
- Uses Starlark build language
- Wide range of programming languages and platforms supported
- Built-in caching and distributed builds for faster and efficient builds

Let's have a look

```
cc_library(
  name = "lib",
  srcs = ["lib.cc"],
)

cc_binary(
  name = "main",
  srcs = ["main.cc"],
  deps = [":lib"],
)

cc_test(
  name = "test",
  srcs = ["test.cc"],
  deps = [":lib"],
)
```

Pros:

- Works with a plethora of languages out-of-the-box and can easily be extended
- Remote caching and remote execution support
- Smarter and more reliable caching based on content hashing
- Optimised for speed, reproducibility and portability
- Used by a bunch of tech giants like: Google, Uber, Dropbox, SpaceX, ...

Cons:

- Tooling support could improve
- Quite intimidating for newcomers

Bazel in-depth

Target

A target is something that bazel can build. In its simplest form a target is made of:

- 0..N input files
- 0..N output files
- 0..N actions

Actions

Actions are atomic commands that are executed in a build to generate outputs from a given set of input. A good example is an action running a compiler or a code-generator

Workspaces

A workspace in bazel is simply a directory containing any number of source files and a WORKSPACE file at the top-level path of said workspace.

```
/a: directory
    /WORKSPACE
    /b: directory
        /WORKSPACE
    /c: directory
        /source.c
```

Package

A package in Bazel is a sub-path in a repository containing a BUILD file and any amount of other files. Each package may have any amount of build targets.

Labels

Labels are unique identifiers for targets. A label in bazel follows the following structure:

`@workspace_name//package_name:target_name`

`@` is the current workspace, but it's often times omitted.

BUILD files

`BUILD` (or `BUILD.bazel`) files use macros and rules to instantiate various kinds of build targets

Rules

Allow defining custom reusable logic for build rules

```
def _hello_world_impl(ctx):
    output_file = ctx.outputs.out
    ctx.actions.write(output_file, "Hello World!\n")

hello_world = rule(
    implementation = _hello_world_impl,
    attrs = {},
    outputs = {"out": "%{name}.txt"},
)
```

Macros

Can be used to combine or simplify existing rules

```
def cc_lib_and_binary(name, **kwargs):
  lib_name = "%s.lib" % name

  cc_library(
    name = lib_name,
    **kwargs,
  )

  deps = kwargs.pop("deps", []) + [lib_name]
  cc_binary(
    name = name,
    deps = deps,
    **kwargs
  )
```

Bazel flags and .bazelrc

Bazel provides a huge amount of flags to configure the build and even allows you to define more. `.bazelrc` makes it easier to preset, standardize and categorize flags within a codebase.

```
build --cxxopt="-std=c++14"
build --host_cxxopt="-std=c++14"

build:ci --color=yes
build:ci --curses=yes
build:ci --show_timestamps
build:ci --announce_rc

build:rbe --remote_executor=grpcs://rbe.cluster.engflow.com
```

Useful tools

- Gazelle: a multi-language build file generator
- Buildozer: powerful build editor/rewriter
- Bazelisk: bazel's de-facto version manager
- Exodus: migration tool from maven to bazel developed at Wix
- awesomebazel.com: amazing resource to find out more about the growing bazel ecosystem

Thank You!

- Contact information:
  - Email: antonio@engflow.com
  - LinkedIn: https://uk.linkedin.com/in/antonio-di-stefano-405230108
- Additional resources:
  - Bazel official website: https://bazel.build
  - Bazel GitHub repository: https://github.com/bazelbuild/bazel