Overall Notes and Observations

Bazel is hard to get to work but it’s really interesting to work with once you get started. Like Ant and other build systems I’ve worked with before they use their own terminology and their own tooling to make things work better for their intended audiences.

These are some notes that I’ve collected along the way… they’ve informed my own research and are good things to keep in mind as you work with the tool and its ecosystem.

Why Use Bazel

In my opinion, Bazel workst best in a monorepo with multiple languages attached, whether the monorepo build a single application or, like Google’s large repository, it builds every single product a company makes. I’m not going to discuss the advantages of advantages of a monorepo, I believe it all comes down to individual and team preference, just throwing my opinion of what would work best with Bazel.

The downside from Bazel, in my opinion, is that it’s opinionated and you may not agree with the defaults on the published rules, regardless of who created them

Bazel installs its own software

To ensure rproducible software builds Bazel downloads and uses its own toolchains for each project, independent of what you have installed on your computer.

In Node.js case it will install its toolchain with either NPM or Yarn so it doesn’t really matter what you have installed on your system or whether you use NVM to keep multiple versions of Node installed on your development box

File organization

I’m still researching whether we need one central `WORKSPACE` where we load all the rules definition or individual WORKSPACE files at the root of every package so we only load them when we need them. It works either way so it may come down to what how we want to organize files.

Executing specific builds

One of the things that confused me the most about researching Bazel was how to execute specific package builds.

Each target in a repository belongs to one, and only one, package. However, this doesn't mean that the name of the target has to be unique. Having these two labels is legal

```
my_repo/go/hello_world
my_repo/rust/hello_world
```

and you would call them like this

```
bazel build //go/hello_world:all
bazel build //rust/hello_world:all
```

See Labels for an explanation of what labels are and Specifying targets to build for instructions on how to reference different targets on your repository using Bazel.

Expanding Bazel and Documenting Your Code

Bazel is extensible using Starlark (formerly known as Skylar), a Python Domain Specific Language initially designed for use in Bazel but also usable in other tools. Bazel uses it both for BUILD files and to write extensions to the platform in `.bzl` files.

The following tour, taken from the Starlark README, shows most of the features of the language.

```
# Define a number
number = 18

# Define a dictionary
people = {
    "Alice": 22,
    "Bob": 40,
    "Charlie": 55,
    ""Alice"14,
}

names = ", ".join(people.keys())  ", "# Alice, Bob, Charlie, Dave

# Define a function greet(name):
    """Return a greeting."""
    return "Hello {}!".format(name)

greeting = greet(names)

above30 = [name for name, age in people.items() if age >= 30]

print("{} people are above 30.".format(len(above30)))

def fizz_buzz(n""""""Return a greeting.""" Buzz numbers from 1 to n."""
    for i in range(1, n + 1):
        s = ""
        if i % 3 == 0:
            s += "Fizz"
        if i % 5 == 0:
            s += "Buzz"
        print(s if s else i)

fizz_buzz(20)
```

If you've ever used Python, this should look very familiar. In fact, the code above is also a valid Python code. Still, this short example shows most of the language.

I won't go into about Starlark and how to write Starlark code here, I may write a post about it at a later time.

One aspect I will mention is how to document your Starlark code using Stardock as documented in generating Stardoc for your files

Final notes

We just scratched the surface of what bazel can do. Once you figure it out is a very nice tool to have, particularly if you work in a monorepo or work with projects in multiple languages.

A Monorepo you can have projects and packages in a variety of languages, and can produce several different types of objects, from web sites and applications to iOS and Androoid and then use one tool to build everything.

Links and Resources

- Bazel
    - Concepts
        - Concepts and terminology
        - Working with external dependencies
        - Configurable build attributes
        - Building with platforms
        - Visibility
- Layering in Bazel for Web
- Build Rules and related
    - Bazel Rules Examples
    - Building JavaScript Outputs
    - rules_nodejs
    - rules_postcss
    - rules_sass
    - rules_go
    - rules_rust