Specific languages and toolchains

In the last post we looked at Bazel, what it is, installation on macOS using homebrew, and a Bazel-based project's basic configuration.

This post will look at using Bazel to run Typescript and Node.js builds using Bazel. Future posts will look at styling and backend toolchains

Javascript, Typescript and Node.js

To work directly with Javascript we'll leverage two items

- Every Javascript file is also a legal Typescript file
- We're already looking at implementing a Node.js-based build system that will use Javascript/Typescript

Before Bazel will work we need to install the following Node packages.

```
npm i -D @bazel/protractor \
@bazel/rollup \
@bazel/terser \
@bazel/typescript \
@types/jasmine \
html-insert-assets \
http-server \
protractor \
rollup \
terser \
typescript
```

At its most basic, the following `WORKSPACE` file will load the Bazel rules for Node.js, tell Bazel where to find the project's package.json and use the default version of Node and NPM. **It will not install the packages in package.json.**

```
load("@bazel_tools//tools/build_defs/repo:http.bzl", "http_archive")
```

```
http_archive(
  name = "build_bazel_rules_nodejs",
  sha256 = "121f17d8b421ce72f3376431c3461cd66bfe14de49059edc7bb008d5aebd16
  urls = ["https://github.com/bazelbuild/rules_nodejs/releases/download/2
)

load("@build_bazel_rules_nodejs//:index.bzl", "node_repositories")

node_repositories(package_json = ["//:package.json"])
```

We can then start building our rules in `BUILD.bazel`. This example is taken from Bazel's rules for [Node.js example app]

The first step is to load the resources that we'll use for this particular build file. There are two types of resources we load:

- Included in a bazel_rules package like `@build_bazel_rules_nodejs`
- Included from Node.js like `@npm//@bazel/protractor` or `@npm//http-server`

```
load("@build_bazel_rules_nodejs//:index.bzl", "pkg_web")
load("@npm//@bazel/protractor:index.bzl", "protractor_web_test_suite")
load("@npm//@bazel/rollup:index.bzl", "rollup_bundle")
load("@npm//@bazel/terser:index.bzl", "terser_minified")
load("@npm//@bazel/typescript:index.bzl", "ts_devserver", "ts_project")
load("@npm//html-insert-assets:index.bzl", "html_insert_assets")
load("@npm//http-server:index.bzl", "http_server")
```

We then define a default vissibility for this build, whether other build or artifacts from other builds can see the artifacts from this build. We make it public.

See [visibility] in the Blaze documentation for more information.

```
package(default_visibility = ["//visibility:public"])
```

We fisrt define an _ASSETS private variable containing data that other tasks in the buildfile will use.

```
_ASSETS = [
  ":bundle.min",
  "//styles:base.css",
  "//styles:test.css",
]
```

The remainder of the BUILD file defines the targets to build the application.

Every target has a set of attributes:

- name defines how we reference the target on the command line and from other targets
- `srcs` define the source files that must be compiled to create the artifact for the target
- deps define other targets that must be built before this target and linked into it There are three different types of dependencies
    - Within the same package (MyBinary’s dependency on `:mylib`)
    - A third-party artifact outside of the source hierarchy (mylib’s dependency on `@npm//@types/jasmine` from NPM)

Some targets have additional attributes. For more information about specifics look at the corresponding `@bazel/*` NPM packages or the corresponding Bazel rules packages.

```
ts_project(
  name = "app",
  srcs = ["app.ts"],
)

ts_devserver"app.ts" = "devserver",
  "devserver"# We'll collect all the devmode JS sources from these TypeScr
)

rollup_bundle(
  name = "bundle",
  entry_point = ":app.ts",
  deps = [":app"],
)

terser_minified(
  name = "bundle.min",
  src = ":bundle",
)

# Copy inde":app"# Copy index.html to the output folder, adding <script>
  name = "inject_tags",
  outs = ["index.html"],
  args = [
    "--out=$@",
    "--html=$(execpath :index.tmpl.html)",
    "--roots=$(RULEDIR)",
    "--assets",
  ] + ["$(execpaths %s)" % a for a in _ASSETS],
  data = [":index.tmpl.html"] + _ASSETS,
)

pkg_web(
  name = "package",
  srcs = [":inject_tags"] + _ASSETS,
)

http_server(
  name = "prodserver",
  data = [":package"],
  templated_args = ["package"],
)

# This could "inject_tags"# This could also be `ts_library`onfig-test",
  testonly = 1,
  srcs = ["app.e2e-spec.ts"],
  extends = ["tsconfig.json"],
  deps = [
    "@npm//@types/jasmine",
    "@npm//@types/node",
    "@npm//protractor",
  ],
)

protractor_web_test_suite(
  name = "prodserver_test",
  srcs = ["app.e2e-spec.js"],
  on_prepare = ":protractor.on-prepare.js",
  server = "//:prodserver",
)

protractor_web_test_suite(
  name = "devserver_test",
  srcs = ["app.e2e-spec.js"],
  on_prepare = ":protractor.on-prepare.js",
  server = "//:devserver",
)

# Just a dummy test so tha"app.e2e-spec.ts"# Just a dummy test so that we
sh_test(
  name = "dummy_test",
  srcs = ["dummy_test.sh"],
)
```