BazelCon

Building a great web application development experience with Bazel

Pejman Ghorbanzade (he/him)
Staff Software Engineer, Aurora Innovation

Developer Productivity

Performance

Tools and processes continuously provide **short feedback cycles** so that developers can iterate quickly.

Simplicity

Tools and processes are designed with the right **level of abstraction** so that developers can work effortlessly.

Web Development Ecosystem

- Developer-friendly tools
- Short build and development cycles
- Hot module replacement
- Fast test execution
- Easy Dependency management
- Intuitive deployment process

“Wouldn’t we be better off without Bazel?”

— esteemed co-worker

About Me

Staff Software Engineer at Aurora Innovation

Building tooling to improve developer productivity

8+ years of professional experience

Ex founder of a developer tools startup

Ex Canon Medical Informatics

Ex VMware Carbon Black

About Aurora

Delivering the benefits of self-driving technology, safely, quickly and broadly.

We are hiring!

Diverse and Fast Growing

- 300k+ build targets
- 50k+ CI jobs per day
- 700k+ remote executed actions per week
- 96% cache hit rate
- 40+ web applications

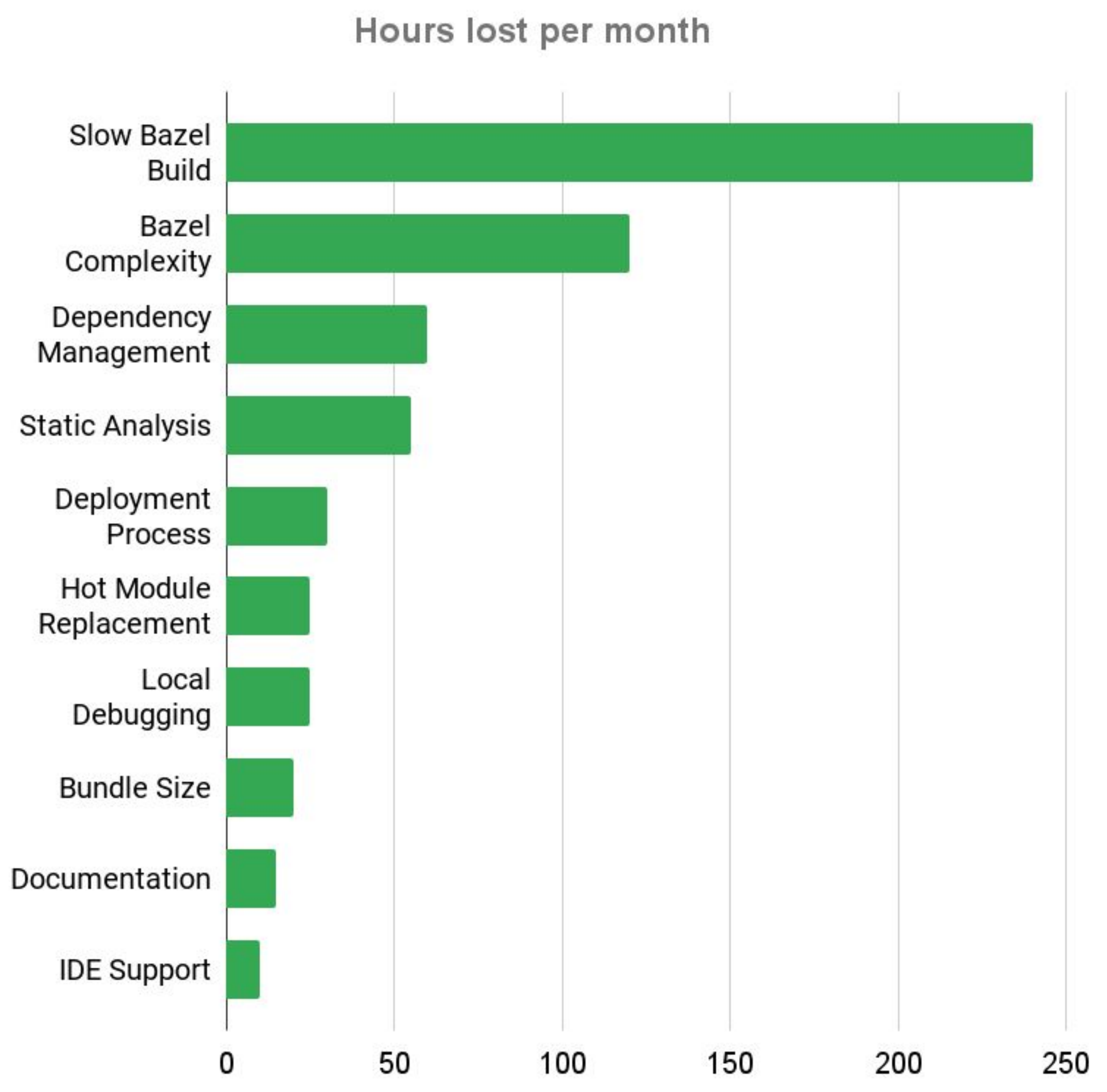
Inconsistency Fosters Complexity

- Build Logic
- Code Quality
- Testing Practices
- Development Style

Developer Productivity Survey

Aurora web developers used to lose

240 hours per month to slow builds.



Performance

Build Performance Profiling

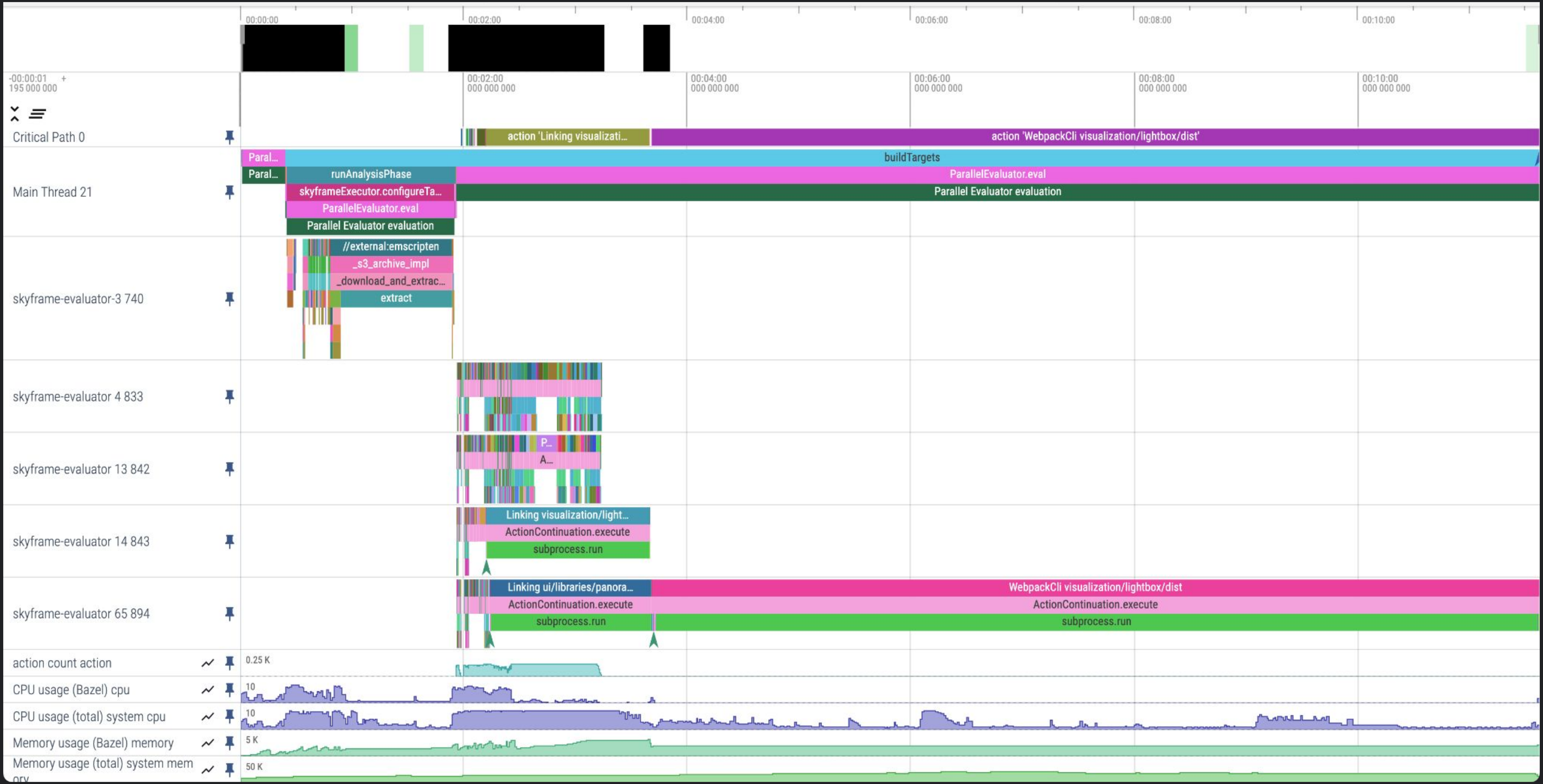
- Using **JSON Trace Profiling** for profiling build time of specific targets
- Using **bazel analyze-profile** for reporting performance of specific build phases
- Using **chrome://tracing** for visualizing build profiles and identifying bottlenecks
- Using internal tools to continuously monitor changes to build performance

```
$ bazel clean --expunge

$ bazel build //my/app:bundle        \
    --generate_json_trace_profile   \
    --profile “my_app.profile.gz”   \
    --noremote_accept_cached        \
    --noslim_profile                \
    --experimental_profile_include_primary_output \
    --experimental_profile_include_target_label
```

```
$ bazel analyze-profile my_app.profile.gz

Launch phase time               1.086 s    0.90%
Init phase time                34.902 s   29.03%
Evaluation phase time           0.961 s    0.80%
Analysis phase time            30.050 s   25.00%
Preparation phase time          0.052 s    0.04%
Execution phase time           53.129 s   44.20%
Finish phase time               0.031 s    0.03%
------------------------------------------------
Total run time                120.213 s  100.00%
```


00:00:00
00:02:00
00:04:00
00:06:00
00:08:00
00:10:00
-00:00:01 195 000 000
00:02:00 000 000 000
00:04:00 000 000 000
00:06:00 000 000 000
00:08:00 000 000 000
00:10:00 000 000 000
Critical Path 0
action 'Linking visualizati…
action 'WebpackCli visualization/lightbox/dist'
Main Thread 21
Paral…
buildTargets
Paral…
runAnalysisPhase
ParallelEvaluator.eval
skyframeExecutor.configureTa…
Parallel Evaluator evaluation
ParallelEvaluator.eval
Parallel Evaluator evaluation
skyframe-evaluator-3 740
//external:emscripten
_s3_archive_impl
_download_and_extrac…
extract
skyframe-evaluator 4 833
skyframe-evaluator 13 842
P…
A…
skyframe-evaluator 14 843
Linking visualization/light…
ActionContinuation.execute
subprocess.run
skyframe-evaluator 65 894
Linking ui/libraries/panora…
WebpackCli visualization/lightbox/dist
ActionContinuation.execute
ActionContinuation.execute
subprocess.run
subprocess.run
action count action
0.25 K
CPU usage (Bazel) cpu
10
CPU usage (total) system cpu
10
Memory usage (Bazel) memory
5 K
Memory usage (total) system mem ory
50 K

Findings

Webpack

Transpiling and type checking TypeScript with TSC and in Webpack is extremely inefficient.

Cache Points

Creating extra cache points significantly improves incremental builds and development server times.

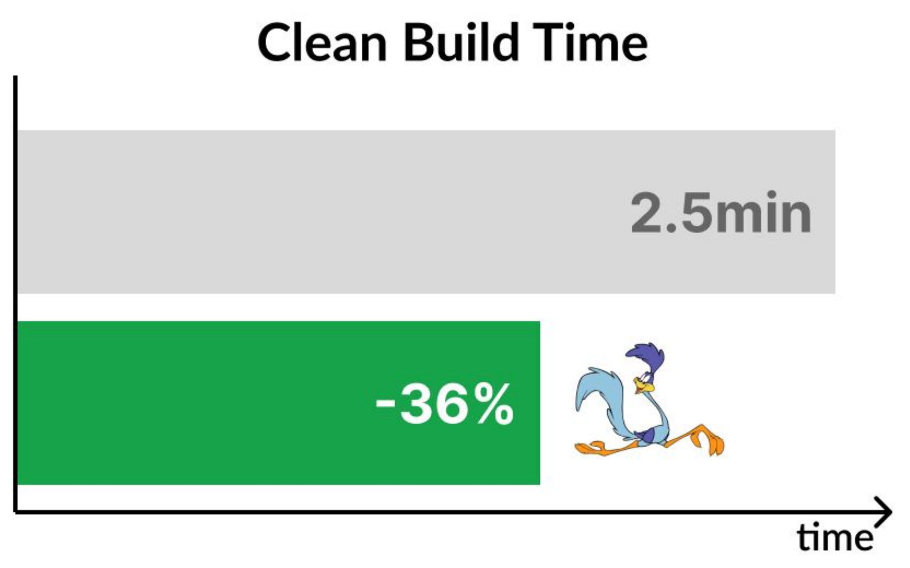
Dependencies

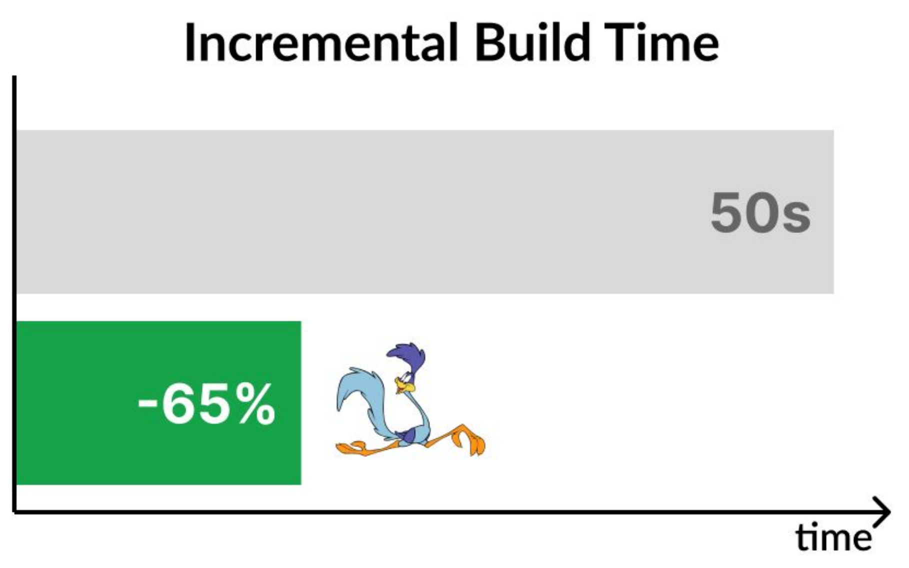
Updating dependencies and reducing bundle size can noticeably improve build performance.

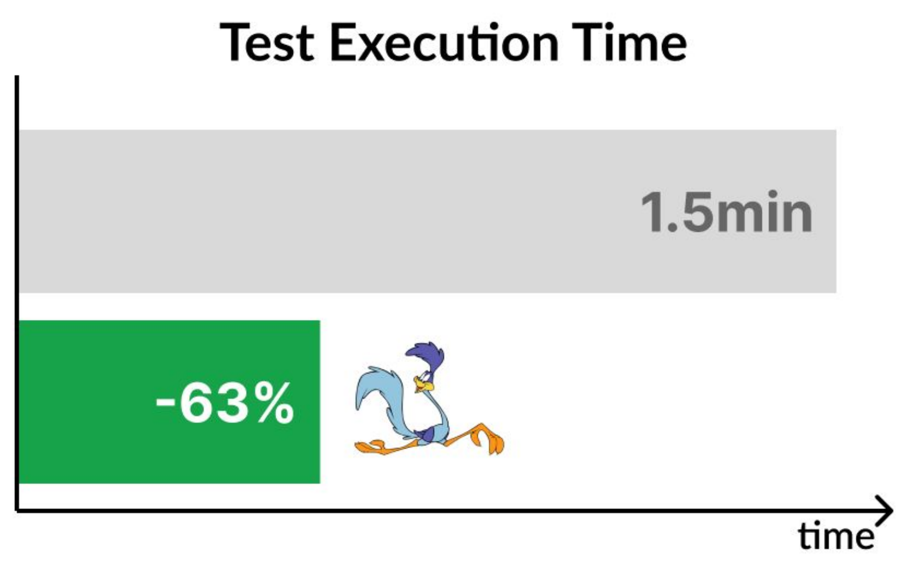
Road Runner

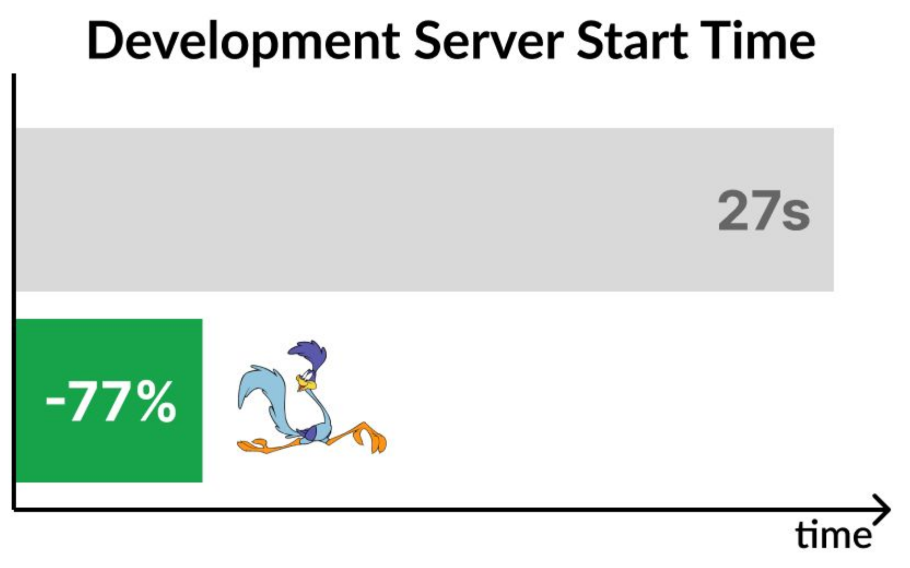
- aspect-build/rules_ts
- aspect-build/rules_swc
- aspect-build/rules_webpack
- swc-loader
- @swc/jest
- minimal BUILD file
- toolchain packages

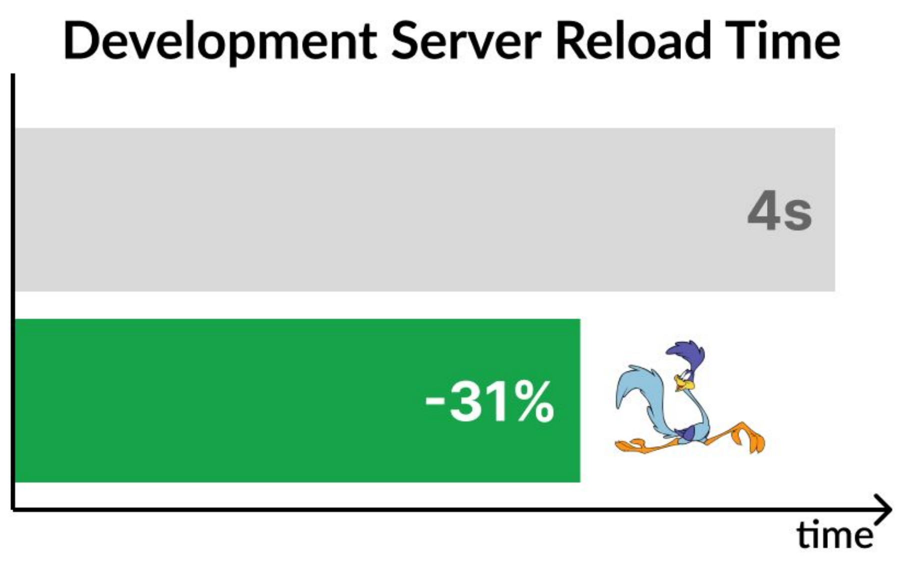
Performance Impact

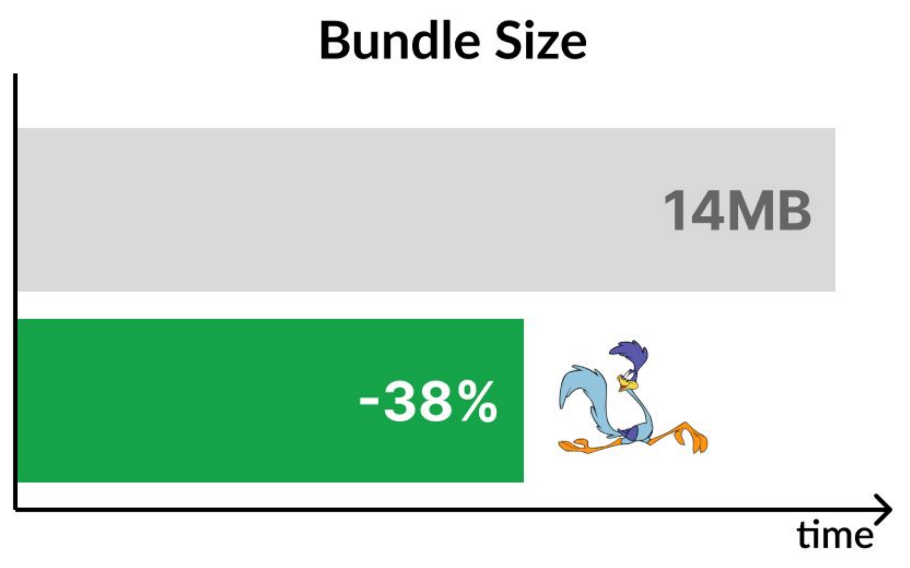












Under The Hood

2x faster build times

- Using ts_project to transpile TypeScript source files with SWC and type-check them with TSC.
- Using rules_webpack to invoke webpack for bundling JavaScript output using swc-loader.

3x shorter feedback cycles

- Passing TypeScript source files to enable hot module replacement using ibazel and swc-loader.
- Using @swc/jest transformer for faster test execution and rules_jest to enable test sharding with Bazel.

rules_ts: ts_project

- Validates tsconfig.json and ensures that dependencies are TsInfo providers.
- Transpiles TypeScript files using TSC or a custom transpiler.
- Performs type-checking using TSC and outside of critical path.

```
ts_project(
   name = "dependencies",
   srcs = srcs,
   deps = deps,
   assets = assets,
   declaration = True,
   extends = "//ts_config_base",
   transpiler = "tsc",
   **kwargs
)
```

rules_swc: transpiling

- Extensible JavaScript transpiler written in Rust and designed for speed.
- Suitable for use with Bazel in numerous short-lived invocations.
- Allows using custom bundlers like Webpack and Rollup.

```
ts_project(
    name = "dependencies",
    srcs = srcs,
    # ...
    transpiler = partial.make(
        swc,
        swcrc = "//:.swcrc"
    ),
    **kwargs
)
```

rules_ts: type checking

- Type checking is incredibly slow relative to transpiling but done outside critical path.
- `ts_project` targets in any dependency tree are type-checked serially.
- `isolatedDeclarations` may enable faster type-checking in the future.
- Most developers could rely on their IDE for type checking during development.

tsconfig.json compiler options

- `isolatedModules`: Allows separate processing of source files for faster transpiling.
- `skipLibCheck`: Allows skipping type checking of declaration files in transitive dependencies.

```json
// tsconfig.json

{
  "compilerOptions": {
    "declaration": true,
    "isolatedModules": true,
    "skipLibCheck": true,
    "strict": true,
    //...
  }
}
```

rules_webpack: webpack_bundle

- Improves remote caching by producing deterministic file hashes and module ids
- Enables hermetic builds by enforcing unique name for produced output

```
webpack_bundle(
    name = name,
    args = args,
    configure_devtool = False,
    configure_mode = False,
    output_dir = True,
    node_modules = "@//:node_modules",
    tags = tags,
    webpack_config = webpack_config,
    **kwargs
)
```

swc-loader Webpack Plugin

- Up to 3x faster build times.
- drop-in replacement for ts-loader
- Can use the same .swcrc configuration file used by ts_project.

```
{
  test: /\.m?js$/,
  exclude: /(node_modules)/,
  use: {
    loader: 'swc-loader',
    options: swcConfig,
  },
}
```

rules_webpack: webpack_devserver

- Up to 3x faster reload times.
- Uses js_run_devserver under the hood.
- Enables hot reloading and hot module replacement using iBazel

```
tags.append("ibazel_notify_changes")

webpack_devserver(
    name = name,
    args = args,
    configure_devtool = False,
    configure_mode = False,
    data = data,
    node_modules = "@//:node_modules",
    tags = tags,
    webpack_config = webpack_config,
    **kwargs
)
```

```ts
// webpack.base.config.ts

module: {
  defaultRules: [
    swcLoaderRule(/\.m?js$/, swcConfig),
    { test: /\.(png|svg|etc)/, type: 'asset/inline' },
    ...(IS_DEV ? [swcLoaderRule(/\.tsx?$/, swcConfig)] : []),
  ],
}
```

@swc/jest

- Up to 5x faster test execution
- *Almost* drop-in replacement for ts-jest
- Different handling of mocking functions
- Does not perform type checking

```
// jest.config.js

module.exports = {
  // ...
  transform: {
    '^.+\\.(j|t)sx?$': '@swc/jest',
  }
};
```

rules_jest

- Supports bazel sharding
- Supports snapshot testing
- Slightly better caching
- Requires Node 18

```
jest_test(
  name = "test",
  config = ":jest.config.js",
  data = [
    "foo.ts",
    "foo.test.ts",
    "bar.test.ts"
  ],
  node_modules = "@//:node_modules",
  shard_count = 2,
)
```

```
$ bazel run :e2e test

Running 4 tests using 4 workers

  ✓  1 [chromium] › example.spec.ts:24:5 › home page (3.1s)
  ✓  2 [chromium] › example.spec.ts:33:5 › log id (8.2s)
  ✓  3 [chromium] › example.spec.ts:9:5 › has title (1.6s)
  ✓  4 [chromium] › example.spec.ts:14:5 › help dialog (7.2s)

  4 passed (9.0s)
```

Simplicity

Design Principles

No Paradigm Shift

Adoption should be seamless without changing everyday development workflow.

Easy Rollout

Adoption should be easy, with minimal risk of introducing regressions.

Minimal Interface

Adoption should involve establishing consistent conventions.

Minimal Interface

- Reducing Friction
- Reducing Complexity
- Reducing Maintenance Cost
- Reducing Migration Cost
- Enforcing Best Practices

```
load("//my/rules:js.bzl", "my_web_app")

my_web_app(
    name = "myapp",
    assets = ["config/.env*"],
    srcs = ["src/**"],
    tests = ["src/tests/**"],
    deps = ["//ui/libraries/mylib"],
)
```

Abstract BUILD file

- Generates ts_project target for transpiling and type-checking
- Generates webpack_bundle target for building web applications for production
- Generates webpack_devserver target for running development server
- Generates jest_test target for running unit tests
- Generates playwright_test target for running end-to-end tests
- Generates other targets for packaging and deployment

Managing Dependencies with package.json

- Meets developers where they are
- Eliminates paradigm shift
- Enforces conventions and best practices
- Facilitates traceability and future upgrades

```json
{
  "name": "myapp",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "clsx": "^2.0.0",
    "nanoid": "^4.0.2",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-icons": "^4.10.1"
  }
}
```

```
generate_package_json_targets(
    name = "my_js_package_json",
    package_json_files = my_package_json_files(),
)

def package_json_dependencies():
    deps = NPM_DEPENDENCIES["//" + native.package_name()]
    if deps == None:
        fail("intuitive error message")
    return deps
```

Abstracting Toolchain Dependencies

- easier rollout of improvements
- easier upgrading of toolchain components
- easier performance impact measurements
- controlled customization points

```python
def my_web_app(name, **kwargs):
  # ...
  deps = kwargs.pop("deps", [])
  deps.append("//tools/build/webpack")

  ts_project(
    name = "dependencies",
    deps = deps,
    # ...
  )
```

Conclusion

Performance

Enabling fast feedback cycles requires choosing build toolchain components that play nicely with Bazel.

Simplicity

Building a great developer experience requires a build process that is intuitive and almost invisible.

BazelCon

Thank you!

pejman.dev / talks / bazelcon23