Optimizing Bazel Sandboxing with a FUSE File System

Overview of Bazel Sandboxing and sandboxfs
By Julio Merino (@jmmv) for FOSDEM on 2020-02-01
What is Bazel?

{Fast, Correct} - Choose two

Build and test software of any size, quickly and reliably 🥰

GET BAZEL
GET STARTED

Why Bazel?

Speed up your builds and tests
Bazel only rebuilds what is necessary. With advanced local and distributed caching, optimized dependency analysis and parallel execution, you get fast and incremental builds.

One tool, multiple languages
Build and test Java, C++, Android, iOS, Go and a wide variety of other language platforms. Bazel runs on Windows, macOS, and Linux.

Scalable
Bazel helps you scale your organization, codebase and Continuous Integration system. It handles codebases of any size, in multiple repositories or a huge monorepo.

Extensible to your needs
Easily add support for new languages and platforms with Bazel’s familiar extension language. Share and re-use language rules written by the growing Bazel community.

https://bazel.build/
What is an Action in Bazel?

Bazel In-Memory Data Structures

**Inputs**
/usr/bin/cc
./parser.h
./parser.c

**Action**
cc -I. -c parser.c -o parser.o

**Outputs**
./parser.o
Why Do We Sandbox Actions?

Bazel In-Memory Data Structures

**Inputs**

- /usr/bin/cc
- ./parser.h
- ./parser.c

**Action**

```
cc -I. -c parser.c -o parser.o
```

**Outputs**

- ./parser.o

---

**Workspace**

- ./lexer.h
- ./lexer.c
- ./parser.h
- ./parser.c

---

File System
Action Sandboxing: Process Isolation

On Linux: user namespaces – On macOS: sandbox-exec

Action Sandboxing: File System Preparation

Bazel In-Memory Data Structures

Inputs
/usr/bin/cc
./parser.h
./parser.c

Action
cc -I. -c parser.c -o parser.o

Outputs
./parser.o

File System

Workspace
./lexer.h
./lexer.c
./parser.h
./parser.c

Sandbox Directory
./parser.h (ro?)
./parser.c (ro?)
./ (rw)

Create sandbox

Workspace
./lexer.h
./lexer.c
./parser.h
./parser.c
./parser.o

Extract outputs
**Action Sandboxing: File System Preparation**

**Bazel In-Memory Data Structures**

**Inputs**
- /usr/bin/cc
- ./parser.h
- ./parser.c

**Action**
cc -I. -c parser.c -o parser.o

**Outputs**
- ./parser.o

**File System**

**Workspace**
- ./lexer.h
- ./lexer.c
- ./parser.h
- ./parser.c

**Sandbox Directory**
- ./parser.h (ro?)
- ./parser.c (ro?)
- ./ (rw)

**Create sandbox**

**Extract outputs**

**One symlink syscall per input**
Scales poorly: 1ms extra per 20,000 actions = 20s extra wall time
sandboxfs: Using FUSE to Avoid Symlinks

Bazel

sandboxfs

Mount Point

CreateSandbox(action1, {
    / → .../scratch/action1 (rw),
    /src/file1.h → .../src/file1.h (ro),
    /src/file1.c → .../src/file1.c (ro),
})
sandboxfs: Using FUSE to Avoid Symlinks

CreateSandbox(action1, {
    / → .../scratch/action1 (rw),
    /src/file1.h → .../src/file1.h (ro),
    /src/file1.c → .../src/file1.c (ro),
})

In-memory operations

action1/ (rw)
action1/src/file1.h (ro)
action1/src/file1.c (ro)
sandboxfs: Using FUSE to Avoid Symlinks

Bazel

CreateSandbox(action1, {
    / → .../scratch/action1 (rw),
    /src/file1.h → .../src/file1.h (ro),
    /src/file1.c → .../src/file1.c (ro),
})

sandboxfs

Mount Point

In-memory operations

action1/ (rw)
action1/src/file1.h (ro)
action1/src/file1.c (ro)

Same idea as:

mount --bind (Linux)
mount -t null (BSD)
sandboxfs: Using FUSE to Avoid Symlinks

Bazel

sandboxfs

Mount Point

CreateSandbox(action1, {
    / → .../scratch/action1 (rw),
    /src/file1.h → .../src/file1.h (ro),
    /src/file1.c → .../src/file1.c (ro),
})

CreateSandbox(action2, {
    / → .../scratch/action2 (rw),
    /gfx/libgfx.a → .../libgfx.a (ro),
    /main.a → .../main.a (ro),
})

In-memory operations

In-memory operations

action1/ (rw)
action1/src/file1.h (ro)
action1/src/file1.c (ro)

No remount!

action1/ (rw)
action1/src/file1.h (ro)
action1/src/file1.c (ro)
action2/ (rw)
action2/gfx/libgfx.a (ro)
action2/main.a (ro)

https://github.com/bazelbuild/sandboxfs/
Performance Results (macOS, circa 2018)

<table>
<thead>
<tr>
<th>Target</th>
<th>Machine</th>
<th>Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bazel</td>
<td>MacBook Pro 2017</td>
<td>581s</td>
</tr>
<tr>
<td>Bazel</td>
<td>Mac Pro 2013</td>
<td>247s</td>
</tr>
<tr>
<td>iOS app</td>
<td>Mac Pro 2013</td>
<td>1235s</td>
</tr>
</tbody>
</table>
# Performance Results (macOS, circa 2018)

<table>
<thead>
<tr>
<th>Target</th>
<th>Machine</th>
<th>Local</th>
<th>Symlinked sandbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bazel</td>
<td>MacBook Pro 2017</td>
<td>581s</td>
<td>621s (+6%)</td>
</tr>
<tr>
<td>Bazel</td>
<td>Mac Pro 2013</td>
<td>247s</td>
<td>265s (+7%)</td>
</tr>
<tr>
<td>iOS app</td>
<td>Mac Pro 2013</td>
<td>1235s</td>
<td>4572s (+270%)</td>
</tr>
</tbody>
</table>
## Performance Results (macOS, circa 2018)

<table>
<thead>
<tr>
<th>Target</th>
<th>Machine</th>
<th>Local</th>
<th>Symlinked sandbox</th>
<th>sandboxfs sandbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bazel</td>
<td>MacBook Pro 2017</td>
<td>581s</td>
<td>621s (+6%)</td>
<td>612s (+5%)</td>
</tr>
<tr>
<td>Bazel</td>
<td>Mac Pro 2013</td>
<td>247s</td>
<td>265s (+7%)</td>
<td>250s (+1%)</td>
</tr>
<tr>
<td>iOS app</td>
<td>Mac Pro 2013</td>
<td>1235s</td>
<td>4572s (+270%)</td>
<td>1922s (+55%)</td>
</tr>
</tbody>
</table>
Rewriting sandboxfs from Go to Rust

Go

- Easy to write (intern project!)
- VSCode has great support
- Hit scalability problems
- Started to become hard to maintain (few annotations in the code)
Rewriting sandboxfs from Go to Rust

**Go**
- Easy to write (intern project!)
- VSCode has great support
- Hit scalability problems
- Started to become hard to maintain (few annotations in the code)

**Rust**
- Harder to write
- VSCode has support but is very slow
- Much more confident in the code
- In translating the old Go code, Rust uncovered many concurrency issues in it

# Rewriting sandboxes from Go to Rust

<table>
<thead>
<tr>
<th>Go</th>
<th>Rust</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easy to write (intern project!)</td>
<td>Harder to write</td>
</tr>
<tr>
<td>VSCode has great support</td>
<td>VSCode has support but is very slow</td>
</tr>
<tr>
<td>Hit scalability problems</td>
<td>Much more confident in the code</td>
</tr>
<tr>
<td>Started to become hard to maintain (few annotations in the code)</td>
<td>In translating the old Go code, Rust uncovered many concurrency issues in it</td>
</tr>
</tbody>
</table>

## Commonalities

- pprof for finding performance issues
- FUSE bindings not first-class

Future Work

- Optimize further
  - Current Bazel ↔ sandboxfs protocol very inefficient
- pkg_comp
- Other sandboxing approaches (Microsoft's BuildXL)
- But... beware of OSXFUSE and kexts on Mac
Thank You!

https://bazel.build/
https://github.com/bazelbuild/sandboxfs/

https://jmmv.dev/
https://twitter.com/jmmv/