Birth of the Bazel

An inside perspective of open sourcing Google's build tool.

Bazel was announced on March 24, 2015, by unveiling **https://bazel.io** and dumping code on **https://github.com/bazelbuild/bazel**. Its development has been somewhat public ever since, but it has never been documented how we got to that point. I was one of the people who started this project, so let me tell you how it happened.

Childhood build traumas

Long before I joined Google, I worked on **LilyPond**, a sort of LaTeX for sheet music. After a brief and violent encounter with the GNU recommended system of autoconf/automake, we built our own set of GNU Makefile templates, which were much better (if I may say so), and I even submitted some patches to GNU Make. Years later, we had a typesetter that was pretty good, but we discovered that few musicians used Linux, and most weren't technical enough to compile it, let alone on MacOS or Windows. This put us in the business of providing binaries on three platforms. We built a **packaging system** to cross-compile LilyPond and its dependencies to OSX and Windows from Linux. What we built was pretty terrible, but so were all the alternatives. Seeing the crud that was out there in the open source world, got me interested in this as a problem.

Google's Build Challenges

Google had its share of build problems too. When I joined in March 2007, the build system had already been rewritten three times, leading to "Google3", the monorepo with granular build targets, but the actual compilation was still powered by GNU Make. A fourth rewrite was underway: it was a new, Java-based system for executing the builds on top of the google3 monorepo. It was called Blaze, and its promise was "{correct, fast}, choose two". I was very excited to see it, and immediately wanted to contribute as a 20% project. During my noogler orientation, I went up to one of the Blaze authors in the NY office, to have him explain to me how it worked (lots of graphs, very confusing; I gave up on contributing).

Developers, developers, developers

Fast-forward to Autumn 2013. I had developed a very successful 20% developer tool (a git wrapper around our Perforce server) which got me a position in the Piper team in Munich, and my promotion to Staff SWE was just approved. My manager told me to not be surprised if my name popped up on some docs, because some folks were trying to put together a team of high-performers for a skunkworks Google Cloud experiment.

For context, Google had by then woken up to the revenue potential of cloud computing. To make this a profitable enterprise, leadership had declared that Tech Infrastructure ("TI", the group building infrastructure underpinning the classical Google: data centers, storage systems, compilers, etc.) should pivot to Cloud to make Google's Cloud endeavour a success. Much like earlier (Google is now Social!) and later (Google is now ML!) edicts, this made all of the engineers up and down the ladder scramble to add some flavor of Cloud to their project plans, lest their team would be pivoted for them. Our organization (DevInfra for Developer Infrastructure) at the time was part of TI, so we embraced this too. Our strategy was: if we ship awesome developer tools, developers will want to use Google Cloud, and then their bosses will pick Google Cloud when purchase orders are signed.

A weak point in this plan was that DevInfra grew as a fully internally focused team. The engineers building Google3 tooling were also its customers. This gave rise to some amazing systems, but it also meant that the organization had no product managers, and had never actually met a Cloud customer. Thus, in the tumult of trying to create something for cloud, many ideas emerged, and one of them was the aforementioned skunkwork idea, roughly speaking:

- Google internal development is amazing
- if we give the internal workflow to external developers, they'll be sold
- doing an experiment to try this out will be easy

Google3's scale and uniformity is very impressive, and gives Google unmatched velocity (for a company its size), but it's only amazing if you take Google's size into account. If you asked a small startup to stop developing on their laptops, and instead run all their development through SSH sessions on a complex, proprietary development stack, they'd mostly likely say no. Also, none of the Google3 systems were designed as multi-tenant, so just bringing up a parallel stack for customers to use is a gigantic undertaking.

On the bright side, the proposal had the important bits sorted out: the tool to be built already had a name ("bolt", because thunderbolts happen in the cloud!) and a suitable codename for the entire project was at the top of the TODO list.

An Idea Takes Shape

One night in November, I was sleepless, irked by this proposed plan. I was ruminating "I can do better than that." So, I got out of bed at 4 AM to write my alternate proposal, which was that Google should open source Blaze. A tool like Blaze was not available publicly, but there was demand for it. We could only get adoption if we open sourced it. Over time, it could drive cloud revenue if we offered a public version of our remote execution system.

The Blaze team was sitting down the hallway. I knew them because I often hung out with them over beers, and that in turn led me to drop by to ask questions (‘premium support’). So, I bounced this idea around with them at first. Later, a few of the managers also got involved.

After several rounds of back and forth with all the people involved, I asked for a review meeting with our then director. Her admin scheduled a 1 hour meeting on January 16, 2014. After some last minute frantic preparations (presenting to the director!), I dialed into the meeting, and discovered that -for reasons unknown- the admin had invited the entire DevInfra team (about 150 people), who were dialing in from all over the planet. A bit daunted, I launched off my slide show. Our request was to fund the effort with 4 people, and we’d have something to show in 6 months. Our director was also surprised at the large audience; after the presentation, most (including myself) left the meeting, and the manager for build made the case again, after which it was approved.

“We choose to [go to the moon], not because it is hard, but because it is easy”

If a project seems impossibly hard, you would be discouraged from starting it. Therefore, the things that we do embark on are the things that we think are easy, and for some reason, I thought this was one of them. The limit of my planning imagination was 4 people (what do you need more people for?). The limit of my planning horizon is about 6 months. It seemed that we could get something done within that time frame, so that’s what we asked for.

Working on open source has a cachet with the engineers, so it was an easy sell to the team. With my managerial experience I can now also see an extra reason why the managers were enthusiastic about the project: even if a project doesn’t quite fit with your existing roadmap, any kind of fashionable new initiative related to your team will be funded with headcount, and typically, with more people you can do more of the things you want, even if you don’t deliver the shiny feature as quickly as promised. Some of them might even move onto different projects altogether.

Joining The Revolution

Since I had opened my big mouth and publicly advocated for this project, I thought it was only fair that I should put my money where my mouth was, so I applied to transfer to the team.

Blaze worked pretty nicely inside Google for its users, and open source is cool, so what’s not to like about the project? Once I started on the team, I discovered how the sausage was made, and how some of the ingredients were the skeletons in the closet. It was much harder than I thought (obviously!). Some of the reasons were transient: I had never really worked in Java, and didn’t realize how painful it is to get things done if you don’t know basic idioms of the language. Some were historical: the blaze source code had accreted a lot of baggage to support whatever the company prioritized over the last 7 years (Haskell support! GWT! Components!), but the team had always been too busy to clean up this gunk, so we had to do it now. Such cleanups exposed an inherent difficulty of working on Blaze: the build system is versioned together with our gigantic monorepo, and you can only refactor anything in Blaze if you clean up all the BUILD files in the monorepo in parallel, which is difficult and tedious.

I was also worried that the project would be killed if we didn’t find a flagship user in time, but finding one is a Catch-22: to guarantee continued staffing of the project, you have to be important (ie. have users), but for someone to commit to using it, they’d want to be sure it wouldn’t be yanked from under them. I tinkered with the build of Android Open Source. The build is large but doesn’t have to work on Windows. In principle, it was a suitable candidate, but it was also way too complicated. Fortunately, at the time, all the tooling for building mobile apps was also terrible in one way or another. In particular, builds for Apple inside Google had to run on Apple hardware, and thus needed a build tool that didn’t depend on Google’s development infrastructure which was exclusively Linux-based. Bazel could provide that. This gave us our 1000-ish internal users, which allayed my fears of premature cancellation.

As the original proposer, I was involved in discussions about project direction: who were our target users (GCP users? AppEngine users? Mobile?), what is our first ruleset to ship (Java, C++, Go?). The team that was trying to decide this was the most senior I had been in so far. It was illustrative to see that experience or engineering level doesn’t give anyone (including myself) a crystal ball to predict the future, but if you put them in a room for an hour, they’ll still spend the entire hour debating.

The Name of the Game

The project discussions also had a lighter side. Blaze was a great name internally, but it was trademarked a hundred times over already, so we couldn’t use it externally. We had long discussions about the name. Did you know that some contenders for the name included “bake”, “please”, and “potato”? Fortunately, one of the engineers had a knack for **writing**. She suggested using an anagram instead and came up with Bazel. Everyone immediately liked it. Our first logo (the letter ‘b’ with a leaf) was made on an online-logo design site, and cost only $100 or so (we did not say we were Google).

Cake day

By the end of 2014, we had something that we could ship. Then it took another 4 months to sort out things around the release (documentation, code exports, process, governance etc.). On March 24, 2015 we then tried to do a “silent” launch. We thought we’d just flip the visibility bit and drink a beer to celebrate. However, our “silent launch” became #1 item **on HN** within 30 minutes of flipping the bit, and we had a busy day responding to comments and Pull-Requests. When you're Google, there is no such thing as a silent launch.

Fun memory: we told the folks at Twitter (who had a similar system called Pants) and Facebook (authors of Buck, another Blaze clone) that we were releasing Blaze. A few weeks after our code drop, on April 1st, the Facebook folks were kind enough to surprise us with a cake delivered simultaneously to the MUC and NYC offices. The Munich one was a blue Sachertorte, the most delicious one I ever ate.

In my end is my beginning

When you are working on releasing an internal tool to the open, cleaning up and disentangling the code so it is OK to publish seems like the light at the end of the tunnel. I was foolishly thinking "once we release, our mission is accomplished". Unfortunately, once the code is out there, you still have most of the earlier problems: What is the project mission? How do you measure success? How can we keep our sanity while we evolve the external code base, the internal one and the depot together? In reality, releasing adds a new set of problems, because now you have to deal with release procedures, bugs, patches, and user support for the fledgling open source project on top.

For me personally, it was all a bit too much, so I focused on other tasks in the team, and eventually I switched to the Gerrit team. The first task in my new team: porting the Gerrit build to Bazel.

It took some time, but eventually Bazel found its way to other companies that struggle with building and testing large, multi-language projects. In retrospect, worrying about adoption was unnecessary. Basing the Apple flavor of our internal build on open-source Bazel seemed like a clever shortcut, but it ended up being a very expensive one, taking years to undo.

An element of the original proposal was for Bazel to generate cloud revenue through a remote execution product. In one of the many reorganizations, Google Cloud leadership reconsidered the “developers, developers, developers” customer acquisition strategy, and as a result, the remote execution product never went beyond a closed-alpha. That omission in turn opened up an opportunity for other companies. There is now a flourishing ecosystem of startups that do Bazel consulting, and some offer commercial remote execution products. Seeing Bazel succeed in the world in this way brings me a bit of personal pride and joy every time I see it.

Getting it out there has also been a good career move. After having left Google, I discovered that Bazel experts are in short supply. I decided to capitalize on my expertise, and re-join some of my former team members, who founded **EngFlow**, the company that builds the number one remote execution system for Bazel and CMake.