Steve Yegge Follow
Aug 14, 2020 · 23 min read · Listen

Dear Google Cloud: Your Deprecation Policy is Killing You

God dammit, I didn't want to blog again. I have so much stuff to do. Blogging takes time and energy and creativity that I could be putting to good use: my novels, my music, my game, and so on. But you get me riled enough, and I have to blog.

Let's get this over with, then.

I'll begin with a small but enlightening story from my early days at Google. For the record, I know I've said some perhaps unkind things about Google lately, because it's frustrating when your corporate alma mater makes incompetent business decisions on the regular. But Google's internal infrastructure is truly extraordinary, and you could argue that there is still none better today. The people who built Google were far better engineers than I will *ever* be, as this anecdote should serve to illustrate.

First a wee bit of background: Google has a storage technology called Bigtable. Bigtable was a remarkable technical achievement, being one of the first (if not *the* first) "infinitely scalable" key-value stores: the beginning of NoSQL, basically. These days Bigtable still holds up well in the rather crowded space of K/V stores, but back in the day (2005) it was breathtakingly cool.

One fun bit of trivia about Bigtable is that they had these internal control-plane entities (as part of the implementation) called tablet servers, which had large indexes, and at some point they became a scaling b 8.1K 64 gtable engineers scratched their heads over how to make it scale, and realized that they could replace the tablet servers

Another cool bit of trivia is that for a time, Bigtables became popular and ubiquitous inside Google and everyone and their dog had one. So at one Friday's TGIF all-hands, Larry Page casually asked in passing, "Why do we have more than one Bigtable? Why isn't there just one?" Because in theory, one Bigtable should have sufficed for all Google's storage needs. Of course they never did migrate to just one, for practical software engineering reasons (e.g. blast radius), but the theory was interesting. One database for the whole universe. *(Side note: Anyone know if Sable does this at Amazon?)*

Anyway, here's my story, to get us started on my rant.

One day, after I'd been working at Google for just over 2 years, I got an email from the Bigtable engineering team. It said something along the lines of:

Dear Steve,

Greetings from the Bigtable team. We wanted to let you know that you are running a very, very old Bigtable binary in the [some data center name] data center. That version is no longer supported, and we would like to work with you to help you upgrade to the latest version.

Please let us know if you can schedule some time to work with us on this.

Best,

Bigtable Team

You get a lot of email at Google, as you can imagine, and when I glanced at this one, this is what I first perceived it to be saying:

*Dear **RECIPIENT**,*

Greetings from Some Team. We wanted to let you know blah blah blah blah blah blah blah. Blah blah blah blah blah blah blah BLAH, and blah blah blah blah immediately.

Please let us know if you can schedule some of your precious time to blah blah blah.

I almost deleted it on the spot, but there was this lingering, nagging feeling that it didn't *quite* feel like a form letter, even though it *obviously* wasn't for me, since I didn't have a Bigtable.

But it was weird.

For the rest of the day, as I was alternating between working and deciding which species of gummy shark to try next in the micro-kitchens, of which there were at least three close enough to hit from my seat with a well-aimed biscuit, I thought about that email with a growing sense of mild anxiety.

They had used my name specifically. And the email had been sent to my email address and nobody else's, and not by cc: or bcc:. The tone was very personal and pointed. Maybe it was some sort of mistake?

Curiosity finally got the better of me, and I went to look at my Borg console in the data center they'd mentioned in the email.

And sure enough, I was running a Bigtable there. Whaaaaat? I looked at its contents, and lo! It was from the Bigtable codelab I'd run back in my first week as a Noogler, in June 2005. The codelab had you fire up a Bigtable so you could programmatically write some values to it, and I had apparently never shut it down afterwards. It was still running there, over 2 years later.

There are several remarkable aspects to this story. One is that running a Bigtable was so inconsequential to Google's scale that it took 2 years before anyone even noticed it, and even then, only because the version was old. As a point of comparison, I considered using Google Cloud Bigtable for my online game, but it cost (at the time) an estimated $16,000/year for an *empty* Bigtable on GCP. I'm not saying they're gouging you, but in my own personal opinion, that feels like a lot of money for an empty fucking database.

Another remarkable aspect is that *it was still running after 2 years*. WTF? Data centers come and go; they experience outages, they undergo routine maintenance, they change

And then there is in my opinion the most remarkable aspect of all, which is that an unrelated engineering team in some other state was reaching out to *me*, the owner of some tiny mostly-empty Bigtable instance which had had *zero traffic* for the past 2 years, asking if they could help me upgrade.

I thanked them and shut it down and life went on. But I still think about that letter, thirteen years later. Because I sometimes get similar letters from the Google Cloud Platform. They look like this:

Dear Google Cloud Platform User,

We are writing to remind you that we are sunsetting [Important Service you are using] as of August 2020, after which you will not be able to perform any updates or upgrades on your instances. We encourage you to upgrade to the latest version, which is in Beta, has no documentation, no migration path, and which we have kindly deprecated in advance for you.

We are committed to ensuring that all developers of Google Cloud Platform are minimally disrupted by this change.

Besties Forever,

Google Cloud Platform

But I barely skim them, because what they are really saying is:

*Dear **RECIPIENT**,*

Fuck yooooouuuuuuuu. Fuck you, fuck you, Fuck You. Drop whatever you are doing because it's not important. What is important is OUR time. It's costing us time and money to support our shit, and we're tired of it, so we're not going to support it anymore. So drop your fucking plans and go start digging through our shitty documentation, begging for scraps on forums, and oh by the way, our new shit is COMPLETELY different from the old shit, because well, we fucked that design up pretty bad, heh, but hey, that's YOUR problem, not our problem.


Google Cloud Platform

And the thing is, I get these about once a month. It happens so often and so reliably that I have been inexorably pushed *away* from GCP, towards cloud agnosticism. I no longer take dependencies on their proprietary service offerings, because it actually winds up being less DevOps work, on average, to support open-source systems running on bare VMs, than to try to keep up with Google's deprecation treadmill.

Before I return to shitting on Google Cloud Platform, because boyo I am nowhere *near* finished yet, let's go visit how software engineering works in some other domains. Google engineers pride themselves on their software engineering discipline, and that's actually what gets them into trouble. Pride is a trap for the unwary, and it has ensnared many a Google team into thinking that their decisions are always right, and that correctness (by some vague fuzzy definition) is more important than customer focus.

I'm going to pick a few somewhat arbitrary examples from other big software systems, but hopefully you'll be able to start spotting the pattern everywhere; that pattern being: **Backwards compatibility keeps systems alive and relevant for decades**.

Backwards compatibility is a design goal of all successful systems that are designed for *open* use; that is, implemented as open source, and/or guided by open standards. I feel like I'm stating something that's so obvious that we should all be awkwardly embarrassed, but no. It's a political issue, so I need examples.

The first system I'll pick is the oldest: GNU Emacs, which is a sort of hybrid between Windows Notepad, a monolithic-kernel operating system, and the International Space Station. It's a bit tricky to explain, but in a nutshell, Emacs is a platform written in 1976 (yes, almost half a century ago) for writing software to make you more productive, masquerading as a text editor.

I use Emacs every single day. Yes, I'm also using IntelliJ every day, and that has grown into a powerful tooling platform in its own right. But writing software extensions for IntelliJ is a much more ambitious and complex undertaking than writing extensions for Emacs. And

 

once in a while it might require a minor tweak, but it's really quite rare. I'm not aware of anything I've ever written for Emacs (and I've written a lot) that was ever forced into re-architecture.

Emacs does have a deprecation facility called make-obsolete. Emacs' terminology for fundamental software engineering concepts (like what is a "window") are often different from the industry conventions, because Emacs invented them long, long ago. The perils of being before your time: Your names are all wrong. But Emacs does indeed have deprecation, called *obsolescence* in their lingo.

However, the Emacs folks seem to have a different working definition. A different underlying philosophy, if you will.

In the Emacs world (and in many other domains, some of which we'll explore below), when they make an API obsolete, they are basically saying: "You really shouldn't use this approach, because even though it works, it suffers from various deficiencies which we enumerate here. But in the end it's your call."

Whereas in the Google world, deprecation means: "We are breaking our commitments to you." It really does. That's what it ultimately means. It means they are going to force you to do some work, possibly a large amount of rework, *on a regular basis*, as punishment for doing what they told you to do originally — as punishment for listening to their glossy marketing on their website: Better software. Faster! You do *everything* they tell you to do, and you launch your application or service, and then, bang, a year or two later it breaks down.

This is like selling you a used car that they know is going to break down in under 1000 miles.

These are two very, very different philosophical definitions of "deprecation". Google's definition reeks of planned obsolescence. I don't believe that it's *actually* planned obsolescence in the same sense that, say, Apple perpetrates. But Google definitely plans to break your stuff, in a roundabout way. I know because I worked there as a software

anyone has the courage to recommend, in terms of deprecation cycles, is “try to give your customers 6–12 months to upgrade before you drape them over the barrel.”

This is hurting them far more than they realize, and it will continue to hurt them for years to come, because it’s not part of their DNA to care about customers. More on this below.

For the moment, I’m going to make the bold assertion that Emacs is successful in large part, perhaps even *mostly*, because they take backwards compatibility so seriously. In fact, that’s the thesis of this essay. Successful long-lived open systems owe their success to building decades-long micro-communities around *extensions/plugins*, also known as a *marketplace*. I’ve ranted about Platforms before, and how important they are, and how Google has never once in their entire corporate history ever really understood what goes into making a successful open Platform, not counting Android or Chrome.

Actually I should talk about Android briefly, because you’re probably thinking, hey, what about Android?

First, *Android is not Google*. They have almost nothing to do with each other. Android is a company that was purchased by Google in July 2005, and that company has been allowed to run more or less autonomously, and in fact has remained largely intact through the intervening years. Android is an infamously hairy tech stack, and a just-as-infamously prickly organization. As one Googler put it, “One does not simply walk into Android.”

I’ve done my share of ranting about how bad some of Android’s early design decisions have been. Heck, at the time I was doing that ranting, they were busy rolling out shit like Instant Apps, which is now (surprise!) deprecated, and haha on you if you were dumb enough to listen to them when they told you to port all your stuff to Instant Apps.

But there’s a difference here, a material difference, which is that the Android folks actually DO understand how important Platforms are, they go *well out of their way* to prevent breaking your old Android code. In fact, their efforts to keep backward compatibility are so extreme that even I, during my brief stint in Android-land a few years back, found myself trying to convince them to drop support for some of the oldest devices

 

The Android folks take backwards compatibility to almost unimaginable extremes, which piles on massive amounts of legacy technical debt in their systems and toolchains. Oh boy, you should see some of the crazy stuff they have to do in their build system, all in the name of compatibility.

For this, I award Android the coveted You're Not Google award. You really don't want to be Google. They don't know how to build platforms that can last, whereas Android *does* know how to do it. And so Google is being very wise in one respect: letting the Android folks do things their way.

Instant Apps was a pretty dumb idea, though. You know why? Because it required you to *rewrite and re-architect your application!* As if people are just going to up and rewrite 2 million apps. I'm guessing Instant Apps was probably a Googler's idea.

But you see the difference here. Backwards compatibility comes with a steep cost, and Android has chosen to bear the burden of that cost, whereas Google insists that *you*, the paying customer, bear that burden.

You can see Android's commitment to backwards compatibility in their APIs. It's a sure sign, when there are four or five different coexisting subsystems for doing literally the same thing, that underlying it all is a commitment to backwards compatibility. Which in the Platforms world, is synonymous with commitment to your customers, and to your marketplace.

Google's pride in their software engineering hygiene is what gets them into trouble here. They don't like it when there are lots of different ways to do the same thing, with older, less-desirable ways sitting alongside newer fancier ways. It increases the learning curve for newcomers to the system, it increases the burden of supporting the legacy APIs, it slows down new feature velocity, and the worst sin of all: it's ugly. Google is like Lady Ascot in Tim Burton's Alice in Wonderland:

Lady Ascot: Alice, do you know what I fear most?


To explore the tradeoff of Pretty vs Practical, let's take a peek at a third successful platform (after Emacs and Android) and see how it fares: Java itself.

Java has *tons* of deprecated APIs. Deprecation is super popular with Java programmers, more so than for most programming languages. Java itself, the core language and libraries, deprecates APIs all the time.

To take just one of thousands of examples, killing threads is deprecated. It's been deprecated since Java 1.2, released in December 1998. It's been 22 years since they deprecated it.

My live production code still kills threads *every day*. Is that a good thing? Absolutely! I mean, obviously if I were to rewrite the code today I'd do it differently. But my game code, which has been able to make hundreds of thousands of people happy over the past 2 decades, was written to kill worker threads that take too long, and *I've never had to change it*. I know my system best, and I have literally 25 years of experience with running it in production, and I can tell you: In my use case, killing these particular worker threads is *harmless*. It is not worth it to focus my time and energy on rewriting that code, and praise be unto Larry Ellison (I guess), since Oracle has never made me rewrite it.

I guess Oracle understands Platforms too. Go figure.

You can see evidence all through Java's core APIs of waves of deprecation, like glacier lines in a canyon. There are easily five or six different keyboard focus managers in Java Swing. In fact it's hard to find a Java API that isn't deprecated. But they all still work! I think the only time the Java core team will actually *remove* an API is if it causes a blatant security problem.

Here's the thing, folks: We software developers are all super busy, and we are also faced with competing alternatives in *every* software domain. At any given time, programmers in language X are looking at language Y as a possible replacement. Oh, you don't believe me? What about Swift? Everyone's migrating *to* Swift, not away from it, right? Oho, how little you know. Businesses are taking a mercenary's accounting of their dual mobile teams

 

productive. There's real money at stake here. Yes, there are trade-offs, but on the other hand, moooooooney.

Let's say hypothetically that Apple was dumb enough to pull a Guido van Rossum, and declare that Swift 6.0 is backwards-incompatible with Swift 5.0, much in the way that Python 3 is incompatible with Python 2.

I probably told this story ten years ago, but about fifteen years ago I was at O'Reilly's Foo Camp with Guido, sitting in a tent with Paul Graham and a bunch of other at-the-time mucky-mucks, waiting for Larry Page to fly out in his personal helicopter, and Guido was droning on tonelessly in the sweltering heat about "Python 3000", which he had named in honor of the number of years it would take everyone to migrate to it. We kept asking him why he was breaking compatibility, and he'd answer, "Unicode". And we'd ask him, hey, if we have to rewrite our code, then what other benefits are we going to see? And he'd answer, "Yooooooooooooouuuuuuuuniiiiiiicooooooode".

If you install the Google Cloud Platform "gcloud" SDK, you'll get this notice:

Dear RECIPIENT,

We would like to remind you that support for Python 2 is deprecated, so fuuuuuuck yooooooooooooooooooouuuuuu

…and so on. The Circle of Life.

But the thing is, every single developer has choices. And if you make them rewrite their code enough times, some of those *other* choices are going to start looking mighty appealing. They're not your hostages, as much as you'd like them to be. They are your guests. Python is still a very popular programming language, to be sure — but *golly* did Python 3(000) create a huge mess for themselves, their communities, and the *users* of their communities' software — one that has been a train-wreck in progress for fifteen years and is still kicking.

burned everyone's house down? It's hard to say, but I can tell you, it hasn't been *good* for Python. It's a huge mess and everyone is miserable.

So let's say Apple pulls a Guido and breaks compatibility. What do you think will happen? Well, maybe 80–90% of the developers will rewrite their software, if they're lucky. Which is the same thing as saying, they're going to lose 10–20% of their user base to some competing language, e.g. Flutter.

Do that a few times, and you've lost half your user base. And like in sports, momentum in the programming world is *everything*. Anyone who shows up on the charts as "lost half their users in the past 5 years" is being flagged as a Big Fat Loser. You don't want to be trending down in the Platforms world. But that's exactly where deprecation — the "removing APIs" kind, not the "warning but permitting" kind — will get you, over time: Trending down. Because every time you shake loose some of your developers, you've (a) lost them for good, because they are angry at you for breaking your contract, and (b) given them to your competitors.

Ironically, I played a role in helping Google become the deprecation-happy prima donnas that they are today, when I built Grok, which is a source-code understanding engine that facilitates automation and tooling on source code itself — similar to an IDE, but as a cloud-based service that stores materialized representations of Google's entire multi-billion-line source graph in a big datastore.

Grok provided Googlers with a powerful foundation for doing automated refactorings across the entire code base (literally all of Google). Grok can figure out not just your upstream dependencies (who you depend on), but also your *downstream* dependencies (who depends on you), so when you change an API, you know everyone you're breaking! So you can make a change and know that every consumer of your API is being updated to the replacement version; in fact, often, via a tool they wrote called Rosie, you can automate it entirely.

This permits Google's code base internally to be almost preternaturally "clean", as they have these automated mice scurrying about the house, *There Will Come Soft Rains* style

And honestly it works pretty well for Google… *internally*. I mean, yes, the Go community within Google does get some good-natured laughs at the expense of the Google Java community over the latter's habit of gratuitous continuous refactoring. If you keep twiddling with something N times, then it implies that not only did you fuck it up N-1 times, but after a while it's pretty clear you've probably fucked it up on the Nth try as well. But by and large, they stay on top of it, and stuff stays "clean".

The problem begins when they take that attitude and try to impose it on their Cloud customers and other API users.

I've walked you a bit through Emacs, Android, and Java; let's look at one last successful long-lived platform: The Web itself. Boy, HTTP sure has gone through a lot of iterations since 1995 when we were all using blink tags and under-construction signs in our handwritten HTML pages.

But it still works! And those pages still work! That's right, folks, browsers are some of the world champions at backwards compatibility. Chrome is another example of a rare Google Platform that has their heads screwed on straight, and, you guessed it, Chrome acts effectively like a separate company from the rest of Google.

I'll also give a shout-out to our friends in the Operating Systems business: Windows, Linux, NOT APPLE FUCK YOU APPLE, FreeBSD, and so on, for doing such a great job of backwards compatibility on their successful platforms. (Apple gets like a C-minus at best, since they break shit all the time for no good reason, but somehow the community papers over it on each release, and so far, containers haven't completely obsoleted OS X… yet.)

But wait, you say. Aren't you comparing apples to oranges, with standalone single-machine software systems like Emacs/JDK/Android/Chrome to multi-machine systems and APIs like Clouds?

Well, I tweeted this yesterday, but as a Larry Wall "sucks/rules"-style yardstick, I searched for "deprecated" on Google and Amazon's developer sites, respectively, and even though AWS has *hundreds* more service offerings than GCP, Google's developer docs mention

 

that I can't do unfair comparisons like "deprecation mentions as a function of number of service offerings".

But after all these years, Google Cloud is still #3 (I still haven't written my "**How to Aim For #2 and Miss**" blog post about this), and according to some internal sources, there's some concern that they may sink to #4 soon.

I don't have a slam-dunk silver-bullet argument for you here, to "prove" my thesis. All I have are the colorful examples I've shared, which I've accumulated over 30 years as a developer. I've alluded to the deeply philosophical nature of this problem; in a sense, it's politicized within the software communities. Some folks believe that platform *developers* should shoulder the costs of compatibility, and others believe that platform *users* (developers themselves) should bear the costs. It's really that simple. And isn't politics always about who has to shoulder costs for shared problems?

So it's political. And there will be angry responses to this rant.

As a *user* of Google Cloud Platform, and also (at Grab) of AWS for 2 years, I can tell you that there's a *world* of difference between the philosophies of Amazon and Google when it comes to priorities. I'm not actively developing on AWS, so I don't have as much of a sense for how often they sunset APIs that they have previously dangled alluringly before unwitting developers. But I have a suspicion it's nowhere *near* as often as happens at Google, and I believe wholeheartedly that this source of constant friction, and frustration, in GCP, is one of the biggest factors holding it back.

I know I haven't gone into a lot of specific details about GCP's deprecations. I can tell you that virtually everything I've used, from networking (legacy to VPC) to storage (Cloud SQL v1 to v2) to Firebase (now Firestore with a totally different API) to App Engine (don't even get me started) to Cloud Endpoints to… I dunno, *everything*, has forced me to rewrite it all after at most 2–3 years, and they *never* automate it for you, and often there is no documented migration path at all. It's just crickets.

And every time, I look over at AWS, and I ask myself what the fuck I'm still doing on GCP.


Google Cloud has a Marketplace in which people can offer their software solutions, and in order to avoid the empty-restaurant effect, they had to populate it with some offerings, so they contracted with a company called Bitnami to create a bunch of "click to deploy" solutions, or perhaps I should write "solutions", because they don't solve a fucking thing. They're just there as checkboxes, as marketing filler, and Google never gave a shit whether any of them worked from Day One. I know the PMs who were driving it and I can assure you, those men do not give a shit.

Take click-to-deploy Percona, for instance. I was getting fed up with Google Cloud SQL's shenanigans, and started looking into setting up my own Percona cluster as an alternative. And for once, Google was going to have done something right, and they were going to save me some time and effort with the click of a button!

Go ahead, I dare you. Follow the link and click the button. Choose "yes" to get all the default parameters and deploy the cluster to your Google Cloud project. Haha, joke's on you; it doesn't work. None of that shit works. It's never tested, starts bit-rotting the minute they roll it out, and it wouldn't surprise me if over half the click-to-deploy "solutions" (now we understand the air quotes) don't work *at all.* It's a completely embarrassing dark alley that you don't want to wander down.

But Google is straight-up *encouraging* you to use it. They want you to *buy* it. It's transactional for them. They don't want to *support* anything. It's not part of Google's DNA. Yes, the engineers support each other, as evidenced by my Bigtable anecdote. But for their customer-facing products and services, they have *always* been ruthless in shutting down any offering that doesn't meet their money bar, even if it has millions of users.

And this presents a real problem for GCP, because that DNA is behind all their Cloud offerings. They aren't committed to supporting anything; it's well-known that they refuse to host (as a managed service) any third-party software until *after* AWS has already done the same thing and built a successful business around it, at which point their customers hold them at gunpoint. But that's the bar, to get Google to support something.

This lack of a support culture, combined with a "let's break it in the name of making it

Google, wake the fuck up. It's 2020. You are still losing. It's time to take a hard look in the mirror and answer for yourselves whether you really want to be in the Cloud business.

If you do, then **stop breaking shit**. You guys are rich. We developers are not. So when it comes to shouldering the burden of compatibility, *you need to pay for it.* Not us.

Because there are at least three other really good Clouds out there. They are beckoning.

And now I'll get back to trying to fix all my broken stuff. Sigh.

Tune in next time!

p.s. An update, after having read some of the discussions (which were great, btw). Firebase is not deprecated, nor are there any plans that I know of. However, they have a nasty threading bug that causes the Java client to stop in App Engine, which one of their engineers helped me with ***while I was at Google****, but they never actually fixed it outright, so I have a crummy workaround to restart my GAE app every day. Four years later! Now they have Firestore, which will take work to migrate to as it's totally different, and the Firebase bug's never gonna be fixed. What can we learn from this? You can get help from them* ***if you work there****. It's frightening that I appear to be the only one using Firebase on GAE, since I'm literally writing fewer than 100 keys in a 100% vanilla app, and it stops working every couple days from an acknowledged bug. What can I tell you, except use it at your own risk. I'm switching to Redis.*

I've also seen some folks more experienced with AWS saying that AWS basically never deprecates/sunsets anything, SimpleDB being a great example. My speculation about AWS not having Google's deprecation disease seems to have been justified.

It was also brought to my attention that 20 days ago, Google's App Engine team broke a critical Go library hosting service by deprecating and killing a GAE app being run by one of the core Go engineers. Egg on face indeed.

Finally, I've heard that Googlers are busy discussing this already, and are by and large agreeing with me (love you guys!)—with the caveat that they seem to think that it's not

Oh yeah, and they really did have different species of gummy sharks in a giant self-serve bin in Building 43 in MTV in 2005, hammerhead being my favorite flavor. Larry & Sergey got rid of all the unhealthy snacks by 2006 though. So at the time of my Bigtable story in 2007, there were indeed no gummy sharks and I'm a big fat liar.

When I looked at Cloud Bigtable 4 years ago (give or take), that was the cost. It seems to have come down a bit, but is still an awful lot for an empty datastore, especially given that my first story shows how inconsequential an empty Bigtable is in their grand scheme.

Sorry for offending all the Apple folks, and for not saying enough nice things about Microsoft, etc. I've read all the threads online and nobody has said anything unfair, in my opinion. I appreciate all the discussion this has generated! But sometimes you have to make a big splash to kick the discussions off, you know?

Thanks for reading.

Update 2, Aug 19 2020: Stripe does it right! https://stripe.com/blog/api-versioning

Update 3, Aug 31 2020: A Google engineer in Cloud Marketplace who happens to be an old friend of mine contacted me to find out why C2D didn't work, and we eventually figured out that it was because I had committed the sin of creating my network a few years ago, and C2D was failing for legacy networks due to a missing subnet parameter in their templates. I guess my advice to prospective GCP users is to make sure you know a lot of people at Google…