00:00 - *Saron Yitbarek*

When the New York City subway first started running in 1904, it was a marvel of the modern age. But ... what happens when today's commuter depends on infrastructure that was designed more than a century ago? Trains are packed and often late. Two billion subway rides take place each year in New York, and nobody's marveling anymore. We are tied to yesterday's crumbling infrastructure, and we have to find smart, new ways to make it work.

00:44 - *Saron Yitbarek*

It used to be that infrastructure projects were these big, concrete things we could see—that subway, for example. And because of that physical presence, it was also pretty obvious when they broke down. Highways crack, telephone poles fall over. We know when those things need fixing. Big efforts are necessary to get our lives in sync with aging infrastructure.

01:12 - *Saron Yitbarek*

But things aren't always so obvious. Today we also have IT infrastructure, server farms humming in isolated fields, fiber optic cables spanning oceans, and software infrastructure, too. Like legacy operation system, or shell scripts that nobody dares to replace. When all that IT infrastructure gets old and creaky, we can't see it for ourselves. And yet, the infrastructure that makes today's development work possible is aging, just like an old subway track. And that can mess with our modern lives. Massive new challenges emerge as today's command line heroes work to make sure we're not being boxed in by the past.

02:02 - *Saron Yitbarek*

This is episode 5 of our season-long journey into the world of programming languages. We're looking at 2 languages that have intimate ties to the infrastructure they were first designed for. COBOL is a language native to mainframe computing, and Go is native to the cloud. They're both deeply influenced by their origins. Understanding that might save tomorrow's developers from ending up like a New Yorker crammed into Penn Station.

02:33 - *Saron Yitbarek*

I'm Saron Yitbarek, and this is season 3 of Command Line Heroes, an original podcast from Red Hat.

02:43 - *Grace Hopper*

So many things ahead that we have to do, but we need tremendous amounts of information, correlated, easy to access. We're only at the beginning.

02:53 - *Saron Yitbarek*

Admiral Grace Hopper pioneered high-level programming languages in the 1940s and 50s. And she was able to make that great leap forward because of the infrastructure of her time, mainframe computers.

03:08 - *Chris Short*

Hi, my name's Chris Short.

03:10 - *Saron Yitbarek*

Chris is a principal product marketing manager at Red Hat, and he's a bit of a history buff, too.

03:17 - *Chris Short*

Admiral Hopper in the 40s made FLOW-MATIC, and she's widely considered the grandmother of COBOL, which was revolutionary at the time. So being able to sit there and say, "Hey, just put it on the mainframe," or, "Hey, just store it on the mainframe."

03:31 - *Saron Yitbarek*

It was a major game changer. Suddenly, you've got this machine-independent language, COBOL, that's native to the mainframe environment. Possibilities started opening up.

03:42 - *Chris Short*

COBOL with mainframes really gave every organization the capability to say instead of having a room full of people with pencils, and paper, and calculators, and slide rules, they could just have half a room with a mainframe in it. And then they could have a few people write some applications in COBOL to do all of the math, and logic, and ledgering that their entire finance team could do. So the team of people that you needed to do your finances became a lot less, just because a lot more of the input could be digital as opposed to all hand jam in manually.

04:17 - *Saron Yitbarek*

If you were one of those new COBOL programmers, it would've felt like you have a job for life. Because the infrastructure that your work was based on, all those mainframes, they weren't going anywhere.

04:30 - *Chris Short*

Moore's Law wasn't around back then, so you could go an entire decade working on the same mainframe, potentially, right? Like you didn't have to worry about the next operating system, or the next type of container orchestrator, or the next thing that comes along and AI, or whatever. You could probably spend your whole career working on COBOL. And you knew you were going to be pretty safe.

04:55 - *Saron Yitbarek*

But, Moore's Law did arrive eventually. New infrastructures showed up, too. And these days, programmers are less likely to learn a half-century old language. But here's the thing, those old mainframes aren't actually gone. And that means our need for COBOL developers hasn't vanished either.

05:17 - *Chris Short*

It's getting a lot harder to find COBOL developers. What ends up happening is these mainframes have been here for 50 years, potentially. And these COBOL developers that still can write good COBOL will get paid exorbitant amount of monies to help with projects, and reorganization of data within mainframes. And that skillset is definitely dying off and becoming a highly lucrative career field if you ... you can definitely make a lot of money writing COBOL nowadays.

05:49 - *Saron Yitbarek*

Especially in the manufacturing and finance industries. You can't outrun all that infrastructure that was laid down decades ago. Legacy code permeates work all around the world. It'd be a huge mistake to ignore that old infrastructure and the languages tied to it.

06:08 - *Chris Short*

With 200 billion lines of code laying around, it's going to be really hard to refactor all that. No, I don't think we'll ever see it disappear in our lifetimes, for sure.

06:21 - *Saron Yitbarek*

Chris Short is a principal product marketing manager at Red Hat.

06:28 - *Saron Yitbarek*

I want to drive Chris's point home for a sec. Consider this. COBOL is baked into 95% of all ATM transactions. That's how tied we are to this language. And yet, the average COBOL programmer isn't much younger than the language itself. They are 45, maybe 55 years old. The newbies aren't interested. Which is why I want to introduce you to someone.

06:56 - *Ritika Trikha*

Hi, my name is Ritika Trikha.

06:59 - *Saron Yitbarek*

Ritika's a technology writer, formerly with HackerRank. And she's fascinated by this question of COBOL, and the assumption people make that it's a kind of pointless leftover from the mainframe days.

07:12 - *Ritika Trikha*

Developers today are really not thinking about COBOL, it's out of sight, out of mind.

07:17 - *Saron Yitbarek*

But that could be a recipe for disaster.

07:21 - *Ritika Trikha*

There's a huge volume of COBOL lines of code that are still powering businesses today. At least 1.5 billion new lines of code in COBOL every single year. And I think when you look at the specific industries, it's really interesting. Like there's 50 million lines of code at the IRS. There's 60 million lines of code at the Social Security Administration. And so these businesses and entities are handling some of the most sensitive, important information today, and if we don't continue to power and maintain these mainframes, it could be really destructive.

08:04 - *Saron Yitbarek*

So if we can't escape our old infrastructure, and we can't wave a magic wand to rebuild the whole mainframe universe, what do we do? How do coders, who sometimes only think about the future, start coming to terms with the past? We need to start by facing the problem head on.

08:25 - *Ritika Trikha*

You know, younger generations are going to have to pick up these skills. Or, there has to be some sort of modernization of these mainframes. Either way, this problem isn't going to go away. That's why COBOL is relevant.

08:35 - *Saron Yitbarek*

It's not going to be easy. Ritika figures we've ignored the problem for too long already.

08:42 - *Ritika Trikha*

It's incredibly expensive, hard, and the risk is incredibly high to replace billions of lines of COBOL. It's mission-critical code like Social Security and financial information. And COBOL was specifically designed for these types of large volumes of transactions. So it was designed for business transactions by Grace Hopper in the 60s. And “if it's not broken, why try to fix it” has been the mentality since the 60s, and now we're at a point where we just have decades of very valuable, high volumes of data running on COBOL.

09:22 - *Saron Yitbarek*

In a way, Ritika's calling for a cultural shift. A change in attitude about what's in and what's out. As the world of development starts to actually gain a deeper and deeper past, we have to become more in touch with our own history. You can't escape the aging infrastructure. And that means you can't ignore the history of languages either.

09:47 - *Ritika Trikha*

Something has to be done. When I was at HackerRank, I saw firsthand how many banks and financial institutions are hurting, and desperate almost, for COBOL developers. It's not a problem that's going to go away, and I think either there has to be some sort of modernization of the systems, or we need to keep training folks and incentivizing it. I personally think there's going to be a day where COBOL is actually in again. Really, what's going to happen when all of the developers with COBOL knowledge retire, and no new younger generations of developers are learning COBOL? Something has to give, right? So there needs to be more of a systematic and institutionalized change when it comes to shifting away from COBOL and into the new cloud-based infrastructures.

10:37 - *Saron Yitbarek*

Ritika Trikha is a technology writer based in San Francisco.

10:49 - *Saron Yitbarek*

So what about those cloud-based infrastructures Ritika mentioned? Are the infrastructures we're building today going to chain future generations to particular languages, the way we're still tied to COBOL? Amazon Web Services (AWS) may be the biggest single piece of cloud infrastructure, launched in 2006. Google Cloud Platform arrived in 2008, and Microsoft Azure started in 2010. The Go language, with its focus on concurrency, was made to thrive inside all that new cloud infrastructure. It's a language of its time.

11:26 - *Carmen Andoh*

Hi, my name is Carmen Andoh, and I am a program manager for the Go team at Google.

11:34 - *Saron Yitbarek*

Carmen has an insider's understanding of how Go is tied to today's infrastructure. It starts with the creators of Go having some strong ties to the history of languages.

11:47 - *Carmen Andoh*

Robert Pike, Robert Griesemer, and Ken Thompson. Those names have kind of come through ever since the 1960s. So Ken Thompson invented the programming language B, and then he would go on to invent the UNIX operating system on a summer off. And Rob Pike invented UTF-8, which is a string in coding. He also invented ASCII. He helped co-author the UNIX programming environment. So these two had been coworkers for a very, very long time, and they had been looking at and inventing operating systems in previous programming languages, including C, which Ken Thompson would eventually help write with Dennis Ritchie.

12:28 - *Saron Yitbarek*

Once Pike, Griesemer, and Thompson were all working at Google, they discovered a serious problem. Getting concurrency at scale just wasn't happening. People were waiting hours for a bill to compile. They were working in C++, and had to write all these callbacks and event dispatchers. It was 2009, and our infrastructure was changing again. Languages like C++ were becoming less and less in tune with that new reality.

12:59 - *Carmen Andoh*

The problems were being introduced by things like multicore processors, and network systems, and massive computation clusters, and the web programming model. And then, also, just the growth of the industry and the number of programmers which were going into the thousands and the tens-of-thousands by 2010. And so all of the programming languages up until that point were being worked around, rather than addressing things head on.

13:24 - *Saron Yitbarek*

Eventually, you reach a breaking point and something's got to give.

13:30 - *Carmen Andoh*

Hey, we hated C++ and I said, "Well, let's see if we could invent something new."

13:37 - *Saron Yitbarek*

That new language would need to be exquisitely adapted to our latest infrastructure.

13:43 - *Carmen Andoh*

What happened with the cloud, which was starting to come of age in 2005, was that you now no longer had to handle your own computes, you sort of were renting it elsewhere, and you get a distributed system. But what happens in a distributed system, and in a cloud, is that you have problems of concurrent messaging between distributed systems. You need to make sure that you have no problems with asynchronicity. Go is a programming language that is asynchronous by default. Basically this means that every operation you perform, like sending all these different messages to another in the system, it's done without waiting for the other system to respond back to you. So it can handle multiple messages at any given time.

14:28 - *Carmen Andoh*

And that said, cloud computing is distributed. And so Go was developed to address this exact need. Go became, early on, one of the standard ways of doing this kind of distributed computing. And that's why I think that it picked up a lot of the developer mindshare immediately. Go absolutely is the language of cloud infrastructure, both in its design, but also in the ecosystem of all the cloud infrastructure tooling, and building blocks that have sprung up in the last decade.

15:06 - *Saron Yitbarek*

Soon, major applications like Kubernetes were being written in Go. Google also created Go Cloud, an open source library and set of tools that made Go even more attractive. It became clear, this was the language of a brand new ecosystem. It was the language of the cloud. And it definitely didn't hurt that the creators had reputations for developing languages that lasted.

15:33 - *Carmen Andoh*

I think that the rest of the industry said, "Hey, I don't think that this is going to be going away anytime soon," and the inventors of the language also happen to invent languages that are now in their 50th year, or 60th year.

15:47 - *Saron Yitbarek*

Carmen Andoh is a program manager for the Go team at Google.

15:54 - *Saron Yitbarek*

So we have a new language, Go, designed to deliver the concurrency that cloud infrastructure makes necessary. Sounds great. And Go’s designers tend to create languages that last for a good half century. Also great. But my question is, what will that really mean 50 years from now when Go is more like COBOL? What will it mean when the world is teeming with legacy Go code that only older developers understand? Are we going to be prepared for a time when today's cloud infrastructure is aging? Are we learning lessons from COBOL and the world of mainframe that could help us design a better future for Go and the cloud?

16:40 - *Saron Yitbarek*

Luckily, I found exactly the right person to ask all these questions. And that's next.

16:51 - *Saron Yitbarek*

How do we future-proof our languages? We know they're tied to the infrastructure of their day. And we know that new infrastructures are bound to replace the old ones as decades roll by. So what are we doing today to keep things running smoothly tomorrow?

17:10 - *Kelsey Hightower*

I'm Kelsey Hightower, I'm at Google, I'm a developer advocate and I work bringing open technologies and turning them to products on Google Cloud.

17:19 - *Saron Yitbarek*

Kelsey spends a lot of time thinking about the future of programming. I was curious whether one day we're going to end up with another aging group of programmers with these wizard-like skills around Go, the same way we have a shortage of COBOL wizards today. Are we even planning for that long range future? So Kelsey and I sat down to hash it out.

17:42 - *Kelsey Hightower*

... and so forth. But if you think about some of the new challenges today, things like dealing with the internet, the network, you've got multiple users, hundreds of thousands of concurrent users, different collections of machines and architecture types. So given those new use cases, typically you want to have a new language.

For example, JavaScript is for the web, you don't want to retrofit COBOL so that we can start doing web programming with it. So we have hundreds of languages that are out and pretty well established today, and they're all kind of hyper-focused on their sweet spots.

18:15 - *Saron Yitbarek*

So in that case then, do we need to actively push people towards COBOL? If we're developing these new languages for these new problems and they're highly specialized, and COBOL's still sticking around, do we need to encourage folks to pick it up so we can maintain our legacy code?

18:32 - *Kelsey Hightower*

Well, I think that's going to be a challenge for the enterprise, right? So you've invested 10, 20 years in COBOL, and there is no one actively thinking about learning some new COBOL. Or you don't come out of college just like, "I'm going to double-down..."

18:45 - *Saron Yitbarek*

Right.

18:46 - *Kelsey Hightower*

"...on this language that's older than my parents." So in that world, you have to ask yourself, what is the risk of continuing on with COBOL? Is it still relevant going forward? I think it is still relevant for certain types of workloads, but we have to ask ourselves a question, is it time to progress? Is it time to evolve a little bit? So if you still have billions of lines of COBOL, you're in the situation where you're going to have to try to find all the COBOL talent that's remaining and bring them in house, but maybe we start to think about what can other languages learn from COBOL, and incorporate some of that functionality and libraries into other languages.

19:26 - *Saron Yitbarek*

Life after COBOL, that would be an enormous infrastructure project all on its own. To use my New York subway analogy, it'd be like replacing every underground tunnel. So, going forward, I wanted to know whether we could anticipate those issues, and even do our future selves some favors.

19:48 - *Saron Yitbarek*

If we compare the cloud today to mainframes, are we going to end up in the same boat where we have these legacy code bases that are using kind of old but very stable languages, and we have to kind of reach this new point of figuring out if we should move on or stay the same?

20:05 - *Kelsey Hightower*

So the thing that makes the cloud a bit different, it's not from one manufacturer, right? A lot of cloud providers typically bundle up collections of technology so you have your choice of programming language, you have your choice of programming paradigm, whether you want to event-driven, or it's all web services based on [HTTP]. So what that means is you get to choose what you want to program in, and just kind of focus on what gets solved. So data will come in, data will come out, but you choose how you want to process that data.

20:36 - *Kelsey Hightower*

The mainframe typically just kind of had one main interface, right? Like you write this job, and this is how you submit the job, here's how you monitor the job, and here's where it comes out. So that's very limiting of itself. So if you think about some of the newer mainframes, they also support some of the newer technology, so even in the world of mainframe, you start to see the expansion of programming languages you can use to run your jobs.

20:58 - *Kelsey Hightower*

So then we start to ask ourselves, okay, given that I have my new choice, when is it time to move on from this particular programming paradigm? So I think we don't get stuck.

21:08 - *Saron Yitbarek*

Right.

21:08 - *Kelsey Hightower*

But I think it is going to be nice that there's going to be a new machine that's going to be distributed, maybe there's a lower cost of entry, you don't have to buy the whole mainframe to get started. But we still want that ease of use of here's my job, you run it for me, tell me when it's done.

21:24 - *Saron Yitbarek*

Absolutely. Do you see what's happening, or what's happened to COBOL, happening to any of today's languages? Like for example, Go, do you see us struggling to maintain Go and getting folks who want to write Go in 30 years?

21:38 - *Kelsey Hightower*

I think all languages can suffer that fate, right? So if you think about it, Python's been around for a very long time. I think it's, what, close to 20 years if not more. So I think what happens ... and Python's had a resurgence in its usage, right, it's kind of the foundation of language for machine learning.

21:53 - *Saron Yitbarek*

Yep.

21:54 - *Kelsey Hightower*

For libraries like Tensorflow. So if we use just time alone, I think that's probably not the right way to look at it. It's like how relevant is that community? How relevant is that language willing to adapt? And I think what Python did really, really well, it ... that community saw the ability to make other languages easier to use. For example, Tensorflow's a lot of C++ underneath it, so programming in such a language is probably not as user friendly as something like Python. And you could take Python and use it to generate some of the stuff that people are using for, example, Tensorflow. So now that machine learning is hot, people have brung Python into that new space, so guess what? Python continues to be relevant, and will be relevant for some time to come. And the same thing's going to be true for Go. If Go can continue to be relevant, right, it's like at the foundation of many of our infrastructure tools, many of the cloud libraries, it too will remain relevant. So I think it's all about those communities ensuring that they have a place in the future, and when the future shows up, making sure that they have a story there.

22:58 - *Saron Yitbarek*

Yeah. So how do we future-proof our languages? Meaning, how do we intentionally design a language to make it last, and make it relevant 20, 30 years from now?

23:10 - *Kelsey Hightower*

The people that use the language, so this is something that's really unique I think, in the open source space. Now that we've moved away from commercial languages, right, languages used to come from Microsoft, or Sun Microsystems in the case of Java™, and at that point everyone relied on the vendor to do all the heavy lifting about what the language would be able to do, any new improvements in the run time. Now what we see with things like Go, Node.js, Ruby, all of these are community backed and focused runtimes and languages. So anyone can add new libraries, right? There was a new [HTTP] spec, right, [HTTP/2] came out a few years ago and each of the respective communities just had contributors add those particular libraries, and now guess what? All of those languages are now compatible with the future of the ... kind of the web for the most part.

24:01 - *Kelsey Hightower*

So I think it's really now that individuals have more control if they want their language to be relevant for new use cases by just contributing that functionality themselves. So we're not restricted to one or two companies. If the company goes out of business, then maybe the runtime dies with it. We don't have that problem as much anymore.

24:23 - *Saron Yitbarek*

We've said it on this podcast before. The future is open. But it's fascinating to consider how in another couple decades, the past will be open too. They'll be inheriting infrastructure and languages that are able to morph and evolve.

24:39 - *Kelsey Hightower*

Awesome, thanks for having me, and I look forward to what people do and mainframe is still relevant. So we don't call it legacy, these are classic technologies.

24:47 - *Saron Yitbarek*

Ooh, I like that, classic, very nice.

24:51 - *Saron Yitbarek*

Kelsey Hightower is developer advocate at Google.

24:57 - *Saron Yitbarek*

I'm imagining a future that's rich with classic programming languages, along with new languages that haven't even been born yet. That's a future I'm excited for.

25:07 - *Speaker*

Stand clear of the closing doors, please.

25:12 - *Saron Yitbarek*

You know, in 2017 Governor Andrew Cuomo declared a state of emergency about the New York City subway. His government set aside 9 billion dollars to invest in the aging infrastructure. And that should remind us, sooner or later, we have to take care of the systems we inherit. You don't just race onward to whatever comes next. You bring the past with you.

25:37 - *Saron Yitbarek*

n the world of development, we tend to have a bias towards the future. We think our languages are only useful in the moment, when they're the hot new thing. But, as informational infrastructure continues to age, the history of development becomes more and more real. The past, it turns out, isn't past at all. And it's our job to remember that.

26:05 - *Saron Yitbarek*

You can learn more about COBOL, or Go, or any of the languages we're covering this season, by heading over to redhat.com/commandlineheroes. There's a bunch of great bonus material waiting for you.

26:19 - *Saron Yitbarek*

Next episode is all about Bash. We're exploring the origins of shell scripts, and the key to automation.

26:30 - *Saron Yitbarek*

Command Line Heroes is an original podcast from Red Hat. I'm Saron Yitbarek. Until next time, keep on coding.