# THE DEATH OF GIANTS

## WHY GO WILL REPLACE JAVA AND C

ActiveState

# THE DEATH OF GIANTS

ActiveState

From finance to entertainment to hospitality, the threat of disruption looms larger than ever today. No matter which industry you're in, staying relevant and competitive in today's business environment requires the ability to innovate faster, turn ideas into reality more quickly, and achieve more agility.

But when it comes to the development frameworks that you use to enable this digital innovation, there is a good chance that the programming languages on which it relies were created decades in the past. Today, languages like Java and C remain at the top of the list of the most widely used development frameworks.

To be sure, these legacy languages have served businesses well for many years. They are tried-and-true development frameworks that support a wide range of scenarios.

Yet the world of programming has changed significantly in the decades since Java, C and similar languages were born. Newer, more innovative language frameworks have emerged. The venerable languages of old are no longer the best solutions for meeting enterprise application development needs. They were designed for the pre-cloud age, with a very different set of development practices and expectations than those that prevail today.

The norms of software delivery and deployment have changed dramatically over the past several decades. Microservices architectures have become key to unlocking application agility. Modern apps are designed to be cloud-native, and to take advantage of the loosely coupled cloud services offered by cloud platforms. They operate across distributed environments and use multiple cores to speed performance. Languages like Java and C were not at all designed with cloud-native app delivery in mind. While it is possible to use these languages to build distributed cloud applications, they are an awkward fit for such workloads.

For enterprises, there is a better path forward: rather than continuing to develop in old-generation languages like Java and C, forward-thinking organizations should adopt Go. Go is a much newer and more nimble language. It was created with the cloud, distributed services and agile development in mind. And it's ideally suited for building microservices.

This guide explains why Go is the ideal enterprise application development solution today. It compares Go's history with that of legacy languages, explains Go's technical advantages, and highlights how and why leading companies have now shifted to Go as their main development solution.

## A BRIEF HISTORY OF JAVA, C AND GO

Comparing the histories of Java, C and Go illuminates just how different the design philosophies and contexts of these languages are.

### Java and C

Java originated in 1991 and became publicly available in 1996. Its broad cross-platform support quickly made it popular with enterprise developers, even though the framework remained largely closed-source until 2006. (And even after that, Java's proprietary history complicates use of the language in certain cases, as when Oracle sued Google in 2012 over allegations of improper use of Java APIs.)

C is even older than Java. It was created in 1972 by developers of the Unix operating system, which was originally written in assembly language. They hacked C together to provide a more abstract development framework for Unix.  Although C was designed with this specific use case in mind, with virtually no forethought regarding other types of applications, it was later adapted for a variety of other purposes because it was one of the only fast, cross-platform languages available to early generations of computer scientists.

More than two decades separate the births of Java and C. At the points when these languages were introduced, concepts such as cloud computing and microservices remained years in the future.

The relationship between business needs and development roadmaps was also very different. Satisfying business goals did not require agile development techniques and continuous delivery as they do today. Instead, slow, delay-prone "waterfall" development practices dominated.

While it is possible to extend Java and C to serve purposes for which they were not designed, or shoehorn them into workflows for which they are a poor fit, overcoming the outmoded design characteristics of these languages will only become more difficult as newer software architectures and software delivery techniques are introduced.

### Go

The history of the Go programming language is entirely different from that of Java and C.

Go was introduced in 2007, just as the cloud computing revolution was beginning. In addition, Service Oriented Architecture (SOA), the predecessor to modern microservices, was in its heyday. Continuous Integration tools were becoming common. In short, the programming landscape for which Go was designed from the outset is the same one in which we exist today.

Incidentally, it's worth noting that one of Go's main creators was Ken Thompson, a Unix pioneer who also happened to be a father of the C programming language. So it's fair to say that Go represents a modern incarnation of the same innovative thought and programming talent that birthed C many years ago.

Thus it's no surprise that Go is fast becoming one of today's premier development languages. While it has not yet surpassed Java and C in overall popularity, Go is enjoying an explosive ascent. Go rose between 2016 and 2017 from 55th to 10th place on the TIOBE ranking of programming languages, according to usage share among developers. As noted below, an increasing number of innovative companies are now turning to Go to bring their software operations into the future, and replace legacy languages that (unlike Go) were not designed for today's cloud-native, microservices-centric age.

## GO'S TECHNICAL ADVANTAGES

Go's features reflect its modern design and ability to solve problems that older languages like Java and C cannot elegantly handle.

Go's standout features include the following:

### Go is an Engineered Language

Rather than being put together quickly to solve a pressing need (as was the case with C) or being developed for internal use by a company and only later extended to the developer community (as happened with Java) Go is an engineered language. It was thoroughly planned and purpose-built from the start to support modern hardware architectures and simplify software maintenance for developers and sysadmins.

### Go is Simple and Elegant

Go is designed to be simple to learn and use, yet deliver maximum power to developers. It offers fast compile times and compiles directly to machine code. It enforces clean, easily readable, and dependable code by requiring strong typing. It is extensively and freely documented online.

All of this matters in enterprise development because most enterprise programming teams are large and consist of engineers with varying skillsets and levels of experience. Issues with steep language learning curves, software build delays, and difficulty interpreting previously written lines of code lead to delays and undercut organizations' ability to deliver continuous value, as users expect.

Go helps enterprises avoid all of these problems with its elegant design.

## Go Offers Built-In Garbage Collection

Traditionally, compiled programming languages like C and C++ do not handle memory automatically. Instead, developers have to write memory management into their applications.

This is not the case with Go. Go performs "garbage collection," meaning it automatically identifies occupied memory space that is no longer needed and frees the space for reuse. Go does this regardless of the underlying hardware architecture on which an application runs.

Garbage collection makes programmers' workflows easier and more efficient because they do not have to spend time implementing memory management when writing an application. And, perhaps most importantly, garbage collection helps to make applications more secure because it lowers the risk that a programming error related to memory management could introduce a security vulnerability into an application.

## Go Offers Native Concurrency

Concurrency means that multiple tasks within an application are executed at the same time, rather than sequentially.

On today's modern hardware, concurrency is a must for maximizing application performance on multi-core CPUs. In addition, modern applications are often distributed across multiple servers, each of which may have several CPUs. This also lends itself to the use of microservices applications, which are small services that interact in order to compose a complete application.

Without concurrency, it is impossible to take full advantage of all of the computing power of modern environments, or to have microservices-based applications perform as intended.

While concurrency can be implemented manually in older programming languages, it is much easier to achieve concurrency using Go. With concurrency built into Go, creating scalable and high performing applications is much simpler than with Java or C.

## Go Applications are Easy to Deploy

Go applications are easy for developers to distribute for users, and for users to run and install. Unlike Java, Go applications do not depend on local interpreters or JVMs to run. Go applications are single binaries. This is fantastic for DevOps, as deployments are simpler than with almost any other language.

In addition, the Get tool in Go makes downloading and installing an application as simple as running a single command. This greatly speeds development workflows, and makes it easy for organizations to host Go applications in a central hub.

Easy application distribution and installation is crucial in today's DevOps-oriented world. The DevOps approach to software development emphasizes consistency in software development pipelines and easy upgrade mechanisms. These features make software production more predictable and stable. They minimize delays, wasted time, and costs. They improve security by reducing the number of components and processes that are required to distribute an application.

And they make it easy for host environments (including public clouds like Amazon Web Services (AWS), Azure and Google Cloud) to provide integrations for Go-based applications because the distribution process for these applications is consistent and has a small footprint.

### Go Simplifies Data Science

While Go has a broad range of applications, data science is emerging as an increasingly important context for Go development. Go helps enterprises by making it easy to integrate algorithms written in Python by a data scientist into a Go microservices application.

At the same time, Go provides excellent application performance without requiring developers to optimize parts of their applications in fast languages like C and C++ (although Go supports such usages if desired). And because of the simple, consistent deployment processes discussed in the preceding section, Go allows data scientists to focus on working with data, rather than on the tedious processes of application setup and configuration.

## GO'S LIMITATIONS

No language is the perfect fit for every situation, of course. It's important for enterprises to recognize Go's limitations in order to ensure that they leverage Go most effectively.

For one, Go's strong typing requirements can be frustrating for programmers accustomed to prototyping quickly in a dynamic language. Go has stronger typing than C to enforce correctness in the application by design. Go is also not an object-oriented programming language. It requires programmers who have been weaned in the object-oriented tradition over the past several decades to take a new approach to code design. For both of these reasons, Go comes with a learning curve for many developers, but far less than languages such as C or C++.

By design, Go lacks support for operator overloading and keyword extensibility. This makes Go somewhat clunky to work with for those developers that rely on those features in other languages. The language designers did not support these features, as they chose simplicity and readability over complexity.

While these issues do not rule out Go as a programming solution for modern workloads, they do mean that Go may not be the best fit for every type of programming job or every developer.

## USING GO?

Despite Go's limitations, the language has become the basis for a very active and healthy open source developer community. As of 2016, more than 700 developers had contributed to development of Go.

In addition, Go's advanced features and ability to simplify complex software development processes have led a number of major developer communities and companies to embrace the framework.

Here are just a few examples of major open source platforms built using Go:

Cloud Foundry. Go powers key parts of Cloud Foundry's microservices platform, including gorouter, which manages connectivity for microservices applications running on Cloud Foundry. CloudFoundry chose Go because it is lightweight, agile and well- suited to the microservices framework that is at the core of Cloud Foundry's architecture.

Docker. The Docker container project, which is revolutionizing the way applications are deployed and run, is written in Go. Docker developers chose Go when they began working on Docker in 2013 because a Go application is "easier to install, easier to test, and easier to adopt," according to Docker programmers. They also like the readability of Go, and that it is a "neutral" language, meaning it does not polarize the developer community for political reasons, as some other languages do.

InfluxDB. As a data storage system designed for extremely fast performance and stability, InfluxDB requires highly efficient and reliable programming, which is why they chose to write it in Go.

Ubuntu Snappy. The goal for Canonical's Snappy package management system for Ubuntu Linux was to allow a developer to distribute their application via a small footprint package in a platform-agnostic way. It's no surprise, then, that, as Canonical developer Rich Spencer notes, "Go seems perfectly suited to writing software in a snappy world."

Go also supports the mission-critical operations of a number of major companies, including:

Dropbox. A significant part of the software framework that supports the Dropbox file-sharing platform, which was originally written in Python, has been migrated to Go. Dropbox developers say they made the switch because Go enabled greater scalability as their infrastructure needs grew.

Netflix. A number of the microservices that power Netflix's video streaming platform are written in Go. Go provides a fast but simple programming solution for creating services that need to be able to scale easily and deliver tremendous performance.

Google. Go was born at Google. Google conceived the language because, as Google Engineer Rob Pike explained, the company needed a better language for creating applications that relied heavily on networking; needed to run on multicore processors; could operate across distributed environments, and so on. The languages available before Go didn't support these needs well.

Uber. To operate efficiently, the Uber ride-hailing service needs to be able to keep track of the locations of cars and users in real time. Uber does this with microservices created using Go. Uber engineers chose Go because of its high throughput and low latency. The Go-based location-lookup microservice delivers the fastest lookups of all of Uber's services.

## CONCLUSION: GO IS THE KEY TO ENTERPRISE DIGITAL TRANSFORMATION

As enterprises evolve and continue to migrate workloads to the cloud and microservices architectures, the applications they run in the cloud also need to evolve. They need to be written in languages that enable enterprise developers to take full advantage of cloud computing environments, distributed processing, and microservices application design.

Go was specifically designed with such use cases in mind. It solves many of today's problems related to developing, deploying and maintaining software, while specifically addressing the needs of building tomorrow's cloud-native software.

Go is microservices-friendly and elegant. It automates tedious tasks like memory management, yet manages to deliver top performance at the same time. Go code is easy to read and write. Go is suited to (and already being used for) a broad array of development needs, from databases to geolocation services and Big Data analytics.

## MAKING GO ENTERPRISE-READY WITH ACTIVESTATE

Because Go is an open source language, enterprises seeking to leverage Go to accelerate their development efforts can benefit from commercial support services for Go that are offered by ActiveState. The Go community has been quick to provide patches and updates, as well as provide support via public forums. However, if Go is to penetrate the enterprise to the same extent as a language like Java, commercial support will be required.

To date, ActiveState, a company with a 20-year history of delivering commercial-grade support for open source languages, is the only vendor that has stepped forward to enable enterprises to embrace Go in order to achieve digital transformation.

ActiveState's backing of Perl when it was first introduced is a key reason it became one of the most ubiquitously deployed open source languages within the enterprise. And if history is any indication, that same level of commercial support will be required in order to de-risk Go as the new, de facto open source language of choice for mission-critical enterprise applications.

**ActiveState Software Inc.**
business-solutions@activestate.com

Phone: **+1.778.786.1100**
Fax: **+1.778.786.1133**

Toll-free in North America:
**1.866.631.4581**

ActiveState®

## ABOUT ACTIVESTATE

ActiveState, the Open Source Languages Company, believes that enterprises gain a competitive advantage when they are able to quickly create, deploy, and efficiently manage software solutions that immediately create business value, but they face many challenges that prevent them from doing so. The Company is uniquely positioned to help address these challenges through our experience with enterprises, people and technology. ActiveState is proven for the enterprise: More than two million developers and 97% of Fortune-1000 companies use ActiveState's end-to-end solutions to develop, distribute, and manage their software applications. Global customers like Bank of America, CA, Cisco, HP, Lockheed Martin and Siemens trust ActiveState to save time, save money, minimize risk, ensure compliance, and reduce time to market.