IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, October 2025
Impact Factor: 7.67

Evaluating Go's Role in the Post-Container Era : Microvms, Serverless, and Beyond : A Comprehensive Review

Dr Pushparani MK[1], Vishal Srinivas Kalikar[2], Hardhik Shetty[3] , Vinodkumar Biradar[4], Likith G[5]
Associate Professor, Department of CSD[1]
UG Scholars, Department of CSD[2-5]
Alvas Institute of Engineering & Technology, Mijar, Karnataka, India
drpushparani@aiet.org.in[1], vishal.srinivas.kalikar@gmail.com[2], hardhikshetty345345@gmail.com[3],
Biradarsvinodkumar@gmail.com[4], Likithgowda1546@gmail.com[5]

Abstract: *Go (Golang) has been one of the most influential programming languages in modern infrastructure development. It played a central role in building container technologies such as Docker and Kubernetes, which changed how applications are deployed and scaled. As cloud computing moves toward MicroVMs and serverless models, the relevance of Go is being re-evaluated. This paper reviews Go's position in this new phase of computing by analyzing its strengths in building lightweight, scalable, and maintainable systems. It discusses Go's integration in MicroVM platforms, its use in serverless runtimes, and how it compares to other languages used in cloud environments. The paper also outlines challenges Go faces, including runtime performance and cold start issues in serverless systems. The study concludes that Go continues to be a reliable and efficient choice for infrastructure software, and its simplicity ensures that it remains adaptable as the cloud ecosystem evolves..*

Keywords: Go (Golang), MicroVM, Serverless, Cloud Computing, Distributed Systems, Concurrency

I. INTRODUCTION

1.1 Background and Significance

Go, also known as Golang, was created at Google in 2009 to address issues faced in building large-scale systems. Its goal was to make programming fast, simple, and reliable without sacrificing performance. Over the last decade, Go has become the foundation of many tools that define today's cloud ecosystem—such as Docker, Kubernetes, Prometheus, and Terraform. These systems popularized the use of containers for deploying applications efficiently across different environments.

However, the computing landscape is now shifting. Containers are being complemented or replaced by lighter and faster solutions such as MicroVMs (Micro Virtual Machines) and serverless platforms. These models reduce startup time, improve isolation, and optimize cost by executing workloads only when needed. This transition raises an important question: Can Go maintain its significance in this new post-container world? This review explores that question by studying Go's technical foundations and its evolving use in emerging infrastructure.

1.2 Objectives

This review paper aims to:

- Examine Go's historical role in containerization and cloud computing.
- Analyze its use in MicroVM and serverless technologies.
- Compare its performance and efficiency with similar languages.
- Identify current challenges and possible improvements for future adoption.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, October 2025
Impact Factor: 7.67

II. FOUNDATIONS OF GO IN CLOUD INFRASTRUCTURE

2.1 Language Design

- Go was designed with three main goals: simplicity, speed, and concurrency.
- It compiles directly to machine code, resulting in fast execution and portable binaries.
- Its concurrency model uses goroutines and channels, allowing developers to run thousands of lightweight threads efficiently.
- Go also provides garbage collection and static typing, ensuring performance while preventing common runtime errors.
- These characteristics make Go well-suited for backend services, command-line tools, and infrastructure software that need to handle multiple operations at once.

2.2 Go and the Container Revolution

When Docker was released in 2013, it was built entirely in Go. Kubernetes, launched shortly after, also used Go for its simplicity and ability to handle parallel workloads.

Go's static binaries allowed tools to run in isolated environments without external dependencies. This portability helped containers gain wide adoption across industries.

As container orchestration became essential for managing distributed systems, Go became the default choice for developers building cloud infrastructure.

III. GO IN THE POST-CONTAINER ERA

3.1 Go in MicroVM-Based Systems

MicroVMs aim to combine the security of virtual machines with the speed of containers. Technologies like AWS Firecracker and Kata Containers use lightweight virtualization to run workloads securely with minimal overhead. Go is used to build orchestration layers, control tools, and monitoring systems for such environments. Its concurrency and efficient I/O handling make it ideal for managing multiple MicroVM instances.

In research comparisons, Go-based MicroVM managers showed improved control-plane responsiveness and lower memory usage than Python or Java equivalents.

3.2 Go in Serverless Architectures

Serverless computing runs code on demand, without the developer managing servers. Frameworks such as OpenFaaS, Knative, and AWS Lambda support Go functions.

Go's compiled nature provides stable performance for longer-running tasks, though startup times are slightly slower compared to interpreted languages like Node.js.

However, Go's ability to handle multiple concurrent executions efficiently makes it valuable for event-driven workloads, API gateways, and microservices.

3.3 Go at the Edge and Beyond

Go's use is extending beyond cloud data centers.

At the edge, its small binaries and efficient concurrency make it suitable for IoT and distributed systems.

The Go compiler now supports WebAssembly (WASM), allowing Go code to run inside browsers or embedded devices.

Projects such as IPFS (InterPlanetary File System) and Ethereum clients (Geth) use Go to build decentralized networks, showing its flexibility beyond cloud environments.

IV. COMPARATIVE REVIEW

When evaluating Go against other prominent programming languages such as Rust and Python, several distinctions become apparent. Go produces compiled static binaries that execute quickly and consistently across platforms, whereas

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, October 2025
Impact Factor: 7.67

Python relies on interpretation and Rust on lower-level compilation with stricter memory controls. In terms of concurrency, Go's goroutines and channels provide an intuitive and lightweight model for parallelism, in contrast to Rust's thread-based asynchronous programming and Python's asyncio event loops. Startup performance in Go is generally fast, though not instantaneous, while Rust achieves comparable speed and Python benefits from immediate interpretation at the cost of execution efficiency.

Memory utilization in Go remains moderate, supported by automatic garbage collection that balances speed with ease of development. Rust achieves greater efficiency through manual memory management but requires higher expertise, while Python's dynamic nature results in a heavier footprint. Go's learning curve is gentler than Rust's, making it more accessible for teams transitioning from scripting or object-oriented environments. Python remains easiest for beginners, but its performance limits make it unsuitable for large-scale, high-concurrency infrastructure.

Overall, Go occupies a practical middle ground between productivity and system-level control. It provides a compelling compromise offering the simplicity of dynamic languages with the efficiency of compiled systems making it especially effective for building reliable and maintainable cloud-native and distributed software.

V. BENEFITS AND LIMITATIONS

5.1 Benefits

- Simplicity: Easy to learn and maintain, leading to faster development cycles.
- Concurrency Efficiency: Handles thousands of simultaneous connections with minimal overhead.
- Cross-Platform Deployment: Static binaries make deployment straightforward across systems.
- Community and Ecosystem: A strong open-source community with libraries for networking, cloud, and APIs.
- Integration with Cloud Platforms: Widely supported by major cloud providers and CNCF projects.

5.2 Limitations

- Cold Start Issues: In serverless functions, Go binaries can take longer to initialize compared to interpreted languages.
- Binary Size: Compiled binaries are larger, which affects lightweight deployments.
- Limited Runtime Control: Developers have less control over memory management compared to C or Rust.
- Generics Maturity: Recently introduced generics still lack advanced features found in other languages.

VI. FUTURE OUTLOOK

The future of Go appears strong in both infrastructure and edge computing.

Its integration with MicroVM frameworks suggests continued adoption in lightweight virtualization and secure multi-tenant systems.

Work is ongoing to reduce Go's runtime latency for serverless platforms and improve compiler optimizations. Efforts in energy-efficient execution and reduced binary sizes will make Go more sustainable for future cloud deployments.

Additionally, the growth of Go-based WASM applications may extend its reach into client-side and embedded environments.

VII. CONCLUSION

Go has evolved from a systems programming language into a foundation for modern cloud and infrastructure technologies. As the industry moves past traditional containerization toward MicroVMs and serverless platforms, Go's simple syntax, concurrency model, and compiled nature keep it relevant.

While it faces some technical challenges, especially in cold start and binary size, continuous development and community contributions are addressing these limitations.

Go's journey reflects a language built not just for speed, but for practical and maintainable software in a fast-changing computing landscape.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, October 2025
Impact Factor: 7.67

REFERENCES

[1]. Pike, R. (2012). *Go at Google: Language Design in the Service of Software Engineering.* Google Research.
[2]. AWS (2018). *Firecracker: Lightweight Virtualization for Serverless Computing.* Amazon Web Services.
[3]. CNCF (2023). *Cloud Native Landscape Report.* Cloud Native Computing Foundation.
[4]. Heller, M. (2021). *Why Go Is the Language of the Cloud.* InfoWorld.
[5]. Google (2024). *Go 1.22 Release Notes.*
OpenFaaS (2023). *Serverless Framework Documentation.*